An Exact Method for Design Exploration of Quantum-dot Cellular Automata

Marcel Walter[1] Robert Wille[2,3] Daniel Große[1,3] Frank Sill Torres[1,4] Rolf Drechsler[1,3]
[1]Group of Computer Architecture, University of Bremen, Germany
[2]Johannes Kepler University Linz, Austria
[3]Cyber Physical Systems, DFKI GmbH, Bremen, Germany
[4]Federal University of Minas Gerais, Brazil
{m_walter, grosse, drechsler}@uni-bremen.de, robert.wille@jku.at, franksill@ufmg.br

***Abstract*—Quantum-dot Cellular Automata (QCA) are an emerging computation technology in which basic states are represented by nanosize particles and logic operations are conducted through corresponding effects such as Coulomb interaction. This allows to overcome physical boundaries of conventional solutions such as CMOS and, hence, constitutes a promising direction for future computing devices. Despite these promises, however, the development of (automatic) design methods for QCAs is still in its infancy. In fact, QCA circuits are mainly designed manually thus far and only few heuristics are available. This frequently leads to unsatisfactory results and generally makes it hard to evaluate the quality of respective QCA designs. In this work, we propose an exact solution for the design of QCA circuits that can be configured e. g. to generate circuits that satisfy certain design objectives and/or physical constraints. For the first time, this allows for design exploration of QCA circuits. Experimental evaluations and case studies demonstrate the benefit of the proposed solution.**

I. Introduction

Quantum-dot Cellular Automata (QCA) have been deemed as a candidate for substituting conventional integrated circuit technologies as these are reaching its physical limits [1], [2]. QCA is a *Field-Coupled Nanotechnology* (FCN) that applies cells of quantum-dots in order to represent and process binary information [3]. Theoretical and experimental results indicate that QCA have the potential to allow for systems with highest processing performance and remarkable low energy dissipation [4], [5].

The promising properties of this technology encouraged numerous works on the design of circuits based on QCA. This led to initial QCA designs realizing arithmetic circuits [6], processors [7], and FPGAs [8]. However, most of these circuits have been realized manually. With increasing complexity and/or physical restrictions to be considered (besides others, e. g. dedicated clocking schemes, the crossings of wires, restricted wire lengths, etc.), methods for design automation of QCA circuits are inevitable.

While initial solutions e. g. for the automatic synthesis of QCA circuits have been proposed, they often only focused on a single design objective. In particular, the realization of the data flow in a QCA circuit has been considered. This strongly depends on the respectively assumed clocking scheme – an architectural characteristic of each QCA design that defines how an external clocking signal is distributed. The works by Bubna et al. [9], Ravichandran et al. [10] and Vankamamidi et al. [11] solely focused on implementations based on an unidirectional data flow. Trindade et al. [12] presented a design solution supporting a clocking scheme allowing for a flexible and bidirectional data flow.

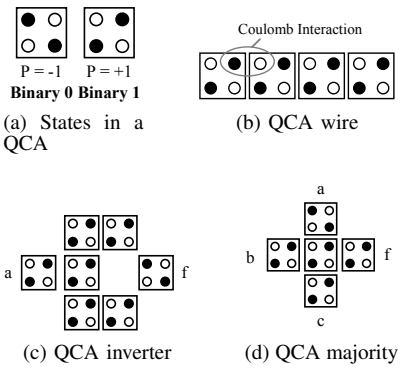
However, beyond that, several further (partially contradictory) objectives have to be considered when designing QCA circuits. For example, whether or not crossings are allowed or whether or not signals are supposed to arrive at the same clock cycle may have a significant impact on the required area. Besides that, physical restrictions such as the minimum distance between wires may have to be considered. The current state-of-the-art mentioned above does not help here, since all solutions proposed thus far (1) rely on *heuristic* algorithms and, hence, cannot guarantee that all objectives/constraints are indeed satisfied and (2) usually focus on a single objective but do not consider alternatives that would allow e. g. to trade-off several objectives. This frequently leads to unsatisfactory results and generally makes it hard to evaluate the quality of respective QCA designs.

In this work, we propose an automatic design method for QCA circuits, which addresses these shortcomings. More precisely, we introduce a symbolic formulation which encapsulates the actual design task (realizing a given function in terms of a QCA circuit) as well as all objectives/constraints that need to be satisfied. Passing the resulting formulation to a reasoning engine allows either for the extraction of a corresponding QCA design realizing the function and satisfying the objectives/constraints or proves that no such realization exists. The symbolic formulation can be configured e. g. to address different objectives or to guarantee the satisfaction of certain physical constraints. This way, an *exact* design method becomes available which is capable of generating different QCA designs satisfying different (complementary) objectives/constraints. For the first time, this allows for a sophisticated design exploration of QCA.

The benefits of the proposed solution are discussed by comparing them with the current state-of-the-art as well as demonstrated on different case studies. This confirms that previously proposed design solutions (based on heuristics) generate results which are far from being optimal. Moreover, the case studies illustrate how the proposed methods, for the first time, allows for the design exploration of QCA circuits with respect to different objectives/constraints.

The remainder of this work is structured as follows: The next section reviews the basics on QCAs as well as the design of the corresponding circuits. Section III introduces the proposed design method, i. e. describes the general idea as well as provides details on the proposed symbolic formulation. Afterwards, the benefits are discussed and demonstrated in Section IV – providing a comparison to previously proposed automatic design methods as well as case studies showing how the solution proposed in this work aids design exploration of QCA circuits. Finally, Section V concludes the paper.

The research reported in this paper has (partially) been supported by the German Research Foundation DFG, as part of Collaborative Research Center (Sonderforschungsbereich) 1320 *EASE – Everyday Activity Science and Engineering*, University of Bremen (http://www.ease-crc.org/). The research was conducted in subproject P04. The research reported here has (partially) been supported by the Brazilian Research Foundations FAPEMIG and CNPq.



(a) States in a QCA (b) QCA wire (c) QCA inverter (d) QCA majority

Fig. 1: QCA states and operations

II. Design of Quantum-dot Cellular Automata

This section reviews the basics on Quantum-dot Cellular Automata and their current design flow. This includes a discussion of the related work and their shortcomings which motivates the contribution proposed in this work.

A. Quantum-dot Cellular Automata

Quantum-dot Cellular Automata (QCA) are a field-coupled technology that conducts computations fundamentally differently from today's technologies. Information is stored and processed by quantum dots, which are structures able to confine an electric charge [4], [1]. A QCA cell is typically composed of four quantum dots arranged at the corners of a square, such as depicted in Fig. 1 (circles represent quantum-dots). Further, each cell contains two free and mobile electrons (illustrated by black dots in Fig. 1), which are able to tunnel between adjacent dots. Tunneling to the outside of the cell is prevented by a potential barrier. The electrons experience mutual repulsion due to Coulomb interaction and tend to locate at opposing diagonals. Consequently, an isolated cell may be in one of two equivalent energy states, called cell polarizations $P = -1$ and $P = +1$ as shown in Fig. 1a. This allows for the codification of binary information by considering that $P = -1$ represents binary 0 and $P = +1$ means binary 1.

Moreover, when placed near to each other, the polarization of one cell influences the polarization of the other – again caused by Coulomb interaction. This causes electrons to avoid to be located in neighboring quantum dots. Exploiting this effect allows for the realization of wires and logic functions.

Example 1. *Based on the concepts introduced above, the following logic elements can easily be realized:*

- *A* wire *as shown in Fig. 1b, where e. g. a 1-state is propagated through several cells by Coulomb interaction (here, from left to right).*
- *An* inverter *as shown in Fig. 1c, where e. g. a 1-state is copied to two paths, which then are combined diagonally, such that the 1-state is inverted to a 0-state (again, from left to right).*
- *A* majority gate *as shown in Fig. 1d, where e. g. a 0-state from input a competes with two 1-states coming from inputs b and c. The output follows the majority of the input states, which is a 1-state in this case.*

Note that, from the majority gate, further logic operations such as OR or AND gate can easily be derived by locking one of its inputs to a 1-state (yielding an OR) or 0-state (yielding an AND).

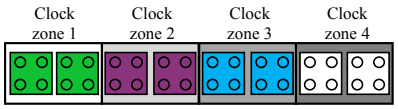


Fig. 2: QCA wire with cells in 4 clock zones

In order to execute these operations, a dedicated clocking is required which, starting with the initialization of the QCA cells, properly propagates the data from cell to cell and avoids metastable states [13]. To this end, an external clock is employed which consists of four phases and regulates the interdot barriers within a QCA cell such that the cell can be polarized or not. In the so-called *release* phase, the interdot barriers are lowered, removing the old polarization of the cell. In the following *relax* phase, the cell remains depolarized. During the third phase, called *switch*, the interdot barriers are raised while a new input is being applied. Consequently, the cell polarizes into one of the two antipodal states. In the following *hold* phase, the cell keeps its polarization and acts as input for adjacent cells in the *switch* phase. Usually, four different clock signals phase-shifted by 90 degree are provided for this purpose [14]. Further, cells are grouped in clock zones, whereby each zone is controlled by one of the four clock signals. The data flow is controlled by placing adjacent clock zones such that the cells which shall pass their data are in the *hold* phase when the cells that shall receive the data are in the *switch* phase. An example illustrates the concepts.

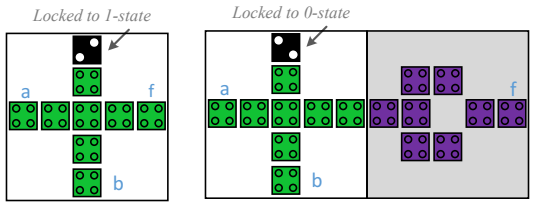
Example 2. *Consider Fig. 2 showing a QCA wire with cells in four different clock zones (emphasized by different coloring). When clock zone 2 is in the* switch *phase, then clock zone 1 is in the* hold *phase. Thus, in this clock phase, cells in clock zone 2 polarize according to the polarization of the adjacent cell in clock zone 1. During the next clock phase, clock zone 2 changes to* hold*, while clock zone 3 is in* switch*. Consequently, data is passed from zone 2 to 3, similar to a pipeline structure.*

B. Design of QCA Circuits

The main goal of each design process in this domain is to realize a given Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ as a QCA circuit. To this end, traditional design solutions for logic synthesis of conventional circuits can be employed in the first steps of the design flow since QCA realizations (denoted QCA *gates* in the following) of typical logic functions such as Inverter, OR, AND, XOR, etc. are readily available (e. g. [15]). However, as discussed above, the resulting gates must be arranged such that the corresponding assignment of underlying clock zones is respected, i. e. data is properly passed from one clock zone to another. To this end, usually a fix arrangement of clock zones is imposed on a QCA layout [11], [9], [16].

Example 3. *Fig. 3 shows a selection of QCA gates for common logic operations (eventually yielding a gate library introduced in [15]). The QCA gates are implemented in a certain amount of clock zones (represented by squares) whereas each clock zone has the size of* 5×5 *QCA cells. Operations such as OR and AND are rather simple and can be realized within one clock zone. In contrast, functions like NAND or NOR require two adjacent clock zones. In this case, it has to be assured that the intended data flow follows the predefined concept of the respectively applied clocking scheme. That means, the QCA structures forming the gates must be arranged such that the output of one QCA structure is placed to an input of another structure which is located in a consecutively numbered clock zone (i. e. 1 is followed by 2, or 4 is followed by 1, etc.).*

There have been several proposals for clocking schemes, like 2DDWave [11], tile-based design [17] or USE (*Universal, Scalable and Efficient*) [16]. Without loss of generality, we employ the latter, which is characterized by a regular architecture and the ability of creating feedback paths, both turning USE in a clocking scheme suitable for design automation.



(a) OR (b) NAND

Fig. 3: Boolean QCA gates realized in one and two clock zones

USE defines a grid of clock zones which are arranged such that all inner clock zones usually have two neighboring clock zones providing data (i. e. possessing a preceding clock zone number) and two neighboring clock zones receiving data (i. e. possessing a consecutive clock zone number). The clock zones are numbered from 1 to 4, whereby consecutive numbered zones have clock signals shifted by 90 degree. Fig. 4a depicts the concept of USE (again, each square is a clock zone of the size of 5×5 QCA cells, following the proposal from [16]). Further, the arrows indicate the possible data flow between adjacent clock zones.

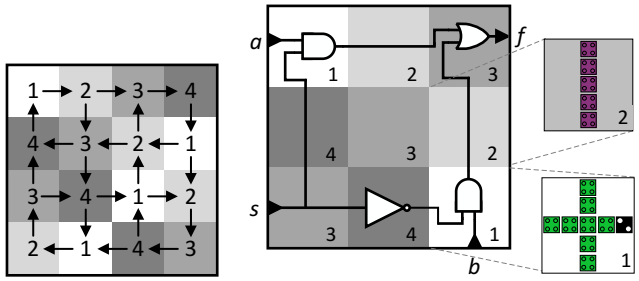
Example 4. *Consider the USE grid shown in Fig. 4a. If clock zones with number 1 are in the* hold *phase, then zones with number 2 are in* switch*, zones with number 3 are in* relax*, and zones with number 4 are in* release*. In this case, data can be transferred from QCA cells in clock zones with number 1 to cells in adjacent clock zones with number 2. In the next step, clock zones with number 2 change to* hold *and clock zones with number 3 to* switch *allowing for data to be transferred from adjacent zones with number 2 to zones with number 3.*

Considering these technology-dependent properties, arbitrary Boolean functions can now be realized by synthesizing the function to a netlist composed of gates supported by the gate library and, afterwards, mapping each gate by the respective QCA gates provided. The mapping itself can be arbitrary as long as consecutive operations are properly connected according to the predefined flow of the USE clocking scheme. Following related work such as [12], the costs of the resulting circuit is measured by different means such as the number of used QCA cells, the area of the resulting grid, the critical path, etc.

Example 5. *Consider the* 2:1 MUX *function* $f = a\bar{s} + bs$ *to be realized. Using conventional design tools yields a gate level representation of this function. Mapping this netlist to a QCA grid as shown in Fig. 4b properly satisfies all clock zone constraints, i. e. all inputs of a QCA gate must come from QCA structures located in a preceding clock zone. In total, the resulting QCA circuit has costs of* 3×3 *clock zones which is equal to* 15×15 *QCA cells. This circuit has a critical path length of 5 clock zones.*

III. Exact Design of QCAs

In this work, we introduce a method for the design of QCA circuits which, in contrast to previously proposed solutions discussed in Section I, is fully automatic (i. e. requires no manual realization) as well as *exact* (i. e. does not rely on heuristics and guarantees the satisfaction of the respectively given design objectives/constraints). In the following, we briefly introduce the main idea of the proposed approach, which relies on a symbolic formulation of the design problem as well as the utilization of reasoning engines. Afterwards, details on the proposed symbolic formulation are provided. Based on that, Section IV illustrates how to apply the resulting solution for QCA design.



(a) USE grid with clock zones (b) Multiplexer mapped to USE grid

Fig. 4: QCA clocking scheme USE

A. General Idea

In order to design QCA circuits, we assume that the given Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$ has already been synthesized using conventional logic synthesis methods and a standard library – yielding a netlist which serves as input as already discussed in the previous section. Besides that, the respective design objectives/constraints are provided, e. g. a USE grid of size $N = x \times y$ onto which the function is supposed to be realized (with x and y being the width and height of the grid, respectively) and/or physical constraints (e. g. the minimum distance between wires, whether crossings are possible, etc.).

Based on that, we formally capture whether the netlist can be realized on the grid adhering the given objectives/constraints. To this end, we define a formulation Φ in unquantified first order logic which symbolically represents the design problem. Passing the resulting symbolic formulation Φ to a reasoning engine (e. g. an SMT solver [18]), either yields an assignment to all variables satisfying all constraints in Φ or proves that no such assignment exists. From this result, either the desired QCA circuit can be extracted or it has been proven no such circuit exists under the respectively given objectives/constraints.

This can be applied for design exploration of QCA circuits e. g. by checking different instances of the formulation with different constraints. For example, enforcing several hard constraints may likely yield an unsatisfiable symbolic formulation (to be checked by the reasoning engine) and, hence, a proof that the function cannot be realized under these constraints. Then, the designer has to decide what constraints shall be loosen. In a similar fashion, minimality with respect to a certain cost metric can be accomplished. For example, if the designer is interested in the minimal grid size onto which a desired function can be realized, he or she simply has to apply a sequence of symbolic formulations with iteratively increasing grid sizes (starting e. g. with a known lower bound). Then, the grid size for which a satisfying assignment can be obtained first, obviously represents the minimum. In a similar fashion, reasoning engines have been utilized e. g. for the exact design of reversible circuits [19], quantum circuits [20] or biochips [21].

B. Proposed Symbolic Formulation

For the symbolic formulation, we separate the elements of the netlist representing the function $f : \mathbb{B}^n \to \mathbb{B}^m$ to be realized into operations and wires.[1] These should be mapped to the formerly mentioned grid of N clock zones, where each clock zone is either empty or holds an element. To express this, a set of Boolean variables g_c^o is used to symbolically represent

[1]Note that we treat fan-outs as operations which perform the identity function of $\mathbb{B}^1 \to \mathbb{B}^2$.

whether clock zone $c \in \mathcal{C}$ is occupied by operation $o \in \mathcal{O}$ ($g_c^o = 1$) or not ($g_c^o = 0$). Additionally, a set of Boolean variables g_c^w is utilized to express whether wire $w \in \mathcal{W}$ is assigned to clock zone c ($g_c^w = 1$) or not ($g_c^w = 0$).

Passing this symbolic formulation to a reasoning engine (without any further constraints) obviously would yield an arbitrary assignment to all variables – symbolically representing an arbitrary QCA circuit, which does not necessarily be compliant to physical or logical constraints. Of course, we are not interested in an arbitrary assignment, though. Hence, we have to restrict the symbolic formulation so that only assignments are allowed that yield valid QCA circuits and satisfy all objectives/constraints.

To this end, we first ensure that each operation is placed exactly once onto the grid. Therefore, we add a constraint which ensures that the number of variables representing operations per clock zone set to true is equal to one. Intuitively speaking, this means that any operation need to be placed somewhere on the grid. More formally:

$$\bigwedge_{o \in \mathcal{O}} \left(\sum_{c \in \mathcal{C}} g_c^o \right) = 1 \tag{C_1}$$

This restriction does not need to hold for wires because they can be of arbitrary length by concatenating numerous wire elements. Nonetheless, it is necessary to enforce that each clock zone is allowed to be occupied at most by either one wire or one operation. To enforce this, we provide another constraint which looks similar to the first one, but is formulated from the perspective of the QCA grid, i. e. for all clock zones, we allow at most one of the operation or wire variables to be set to true:

$$\bigwedge_{c \in \mathcal{C}} \left(\sum_{o \in \mathcal{O}} g_c^o + \sum_{w \in \mathcal{W}} g_c^w \right) \leq 1 \tag{C_2}$$

So far, elements from the netlist can be placed onto the grid, but the interconnections as defined by the netlist are not respected yet. Here, we are faced with two levels: the adjacency relation on the netlist and the adjacency relation on the QCA grid. Both have to match. To formulate this, we introduce the following definitions and notations:

- The USE clocking scheme provides a defined data flow which can be expressed by an adjacency function $\alpha_{\mathcal{C}} : \mathcal{C} \to 2^{\mathcal{C}}$ which returns the set of successor clock zones for a given clock zone. Vice versa, we define an inverse adjacency function $\hat{\alpha}_{\mathcal{C}} : \mathcal{C} \to 2^{\mathcal{C}}$ which returns the set of predecessor clock zones. Similar definitions are available on the netlist, i. e. $\alpha_{\mathcal{O}}$ and $\hat{\alpha}_{\mathcal{O}}$, respectively.
- We use the notation w_{o_1,o_2} to refer to the wire connecting operation o_1 and o_2 in the netlist.
- We will use auxiliary variables $i_{c,c'}$ modeling interconnections between two clock zones on the QCA grid. Their exact meaning is explained later.

We now express the adjacency by two constraints: The constraint C_3 considers the case that an operation has been placed onto some clock zone, and the other constraint C_4 considers the case that a wire segment has been placed onto some clock zone.

More precisely, for C_3 we formalize that the placed operation is either followed by the connected operation directly (so the wire has been skipped) or by a wire segment. In contrast, C_4 states that each segment of a wire is followed by another wire segment or the connected operation as given by the netlist.

Please note, both constraints C_3 and C_4 follow the same scheme: if some operation o or some wire w has been placed on clock zone c, it is implied that some of the adjacent clock zones must be assigned to an adjacent successive operation/wire of the netlist. The corresponding case describing the corresponding predecessor relation is ensured using the inverse adjacency functions.

This results in the following constraints:

$$\bigwedge_{\substack{c \in \mathcal{C}, \\ o \in \mathcal{O}}} g_c^o \implies \left(\bigwedge_{o' \in \alpha_{\mathcal{O}}(o)} \bigvee_{c' \in \alpha_{\mathcal{C}}(c)} \left(g_{c'}^{o'} \vee g_{c'}^{w_{o,o'}} \right) \wedge i_{c,c'} \right) \wedge \left(\bigwedge_{o' \in \hat{\alpha}_{\mathcal{O}}(o)} \bigvee_{c' \in \hat{\alpha}_{\mathcal{C}}(c)} \left(g_{c'}^{o'} \vee g_{c'}^{w_{o',o}} \right) \wedge i_{c',c} \right) \tag{C_3}$$

for the operations, and

$$\bigwedge_{\substack{c \in \mathcal{C}, \\ w \in \mathcal{W}}} g_c^w \implies \left(\bigwedge_{o' \in t(w)} \bigvee_{c' \in \alpha_{\mathcal{C}}(c)} \left(g_{c'}^{o'} \vee g_{c'}^{w} \right) \wedge i_{c,c'} \right) \wedge \left(\bigwedge_{o' \in s(w)} \bigvee_{c' \in \hat{\alpha}_{\mathcal{C}}(c)} \left(g_{c'}^{o'} \vee g_{c'}^{w} \right) \wedge i_{c',c} \right) \tag{C_4}$$

for the wires, where $s(w)$ and $t(w)$ refer to the set of sources and targets of wire w in the netlist, respectively.

We now explain the meaning and functionality of the i-variables: For each adjacency on the USE grid, i. e. each arrow in Fig. 4a leading from a clock zone c to a clock zone c', a new Boolean connection variable $i_{c,c'}$ is introduced which is set to true for each connectivity established by the constraints C_3 and C_4 (i. e. the disjunction can only become true if a connection is established by the left side of the inner conjunction and if the corresponding i-variable is set to true). The i-variables enable us to prevent random wire cycles from appearing on the grid in the next step. Since the underlying netlist is cycle free, we have to ensure the same for the QCA grid. Based on the constraints introduced so far, wire loops without a connection to an operation are possible. To prevent cycles, we introduce an additional set of Boolean variables which we call *path variables* $p_{c,c'}$ for each pair of clock zones $c, c' \in \mathcal{C}$. Note that all N^2 variables are needed here. This is not the case for the interconnection variables where a variable $i_{c,c'}$ has been introduced if and only if c and c' were consecutively numbered clock zones.

The concept of a path variable is as follows: a path variable $p_{c,c'}$ should become true, if there are a number of interconnection variables all set to true, such that there exists a path from c to c'. This is described by two constraints: The first one maps adjacent connections to sub-paths by a single implication. We get:

$$\bigwedge_{\substack{c,c' \in \mathcal{C} \\ c \neq c'}} i_{c,c'} \implies p_{c,c'} \tag{C_5}$$

The second one spans the paths transitively by stating that the existence of a path from clock zone c to clock zone c' and the existence of a path from clock zone c' to clock zone c''

leads to the existence of a path from clock zone c to clock zone c'', i. e.

$$\bigwedge_{\substack{c,c',c''\in\mathcal{C}\\ c\neq c',c'\neq c''}} p_{c,c'} \wedge p_{c',c''} \implies p_{c,c''} \qquad (C_6)$$

Having these preconditions ensured, we are now able to prevent cycles completely by simply disallowing paths that come back to where they started. Formally, no path variable $p_{c,c}$ can be true:

$$\bigwedge_{c\in\mathcal{C}} \overline{p}_{c,c} \qquad (C_7)$$

Passing the resulting formulation to a reasoning engine would lead to almost valid results. Now, it only has to be ensured that the lengths of all fan-in paths to any multi-input operation on the grid do not differ by more than 3 clock zones, exploiting the pipeline-like structure of QCA circuits and therefore leading to highest throughput.

To respect this restriction, we add another constraint to the formulation. Essentially, we express that the number of all clock zones occupied by all fan-in paths to an operation with more than one input ($\mathcal{P}(o)$) need to be the same. Additionally, the clock zone number (cn) of the first path element (p^*) is considered to respect the clock zone constraints correctly (leading to a maximum difference of 3 clock zones). We get:

$$\bigwedge_{o\in\mathcal{O}} \bigwedge_{\substack{p_1,p_2\in\mathcal{P}(o)\\ p_1\neq p_2}} \left(cn(p_1^*) + \sum_{\substack{c\in\mathcal{C}\\ e_1\in p_1}} (g_c^{e_1}) \right) = \left(cn(p_2^*) + \sum_{\substack{c\in\mathcal{C}\\ e_2\in p_2}} (g_c^{e_2}) \right) \qquad (C_8)$$

To complete the symbolic formulation, some "minor constraints" are required. Due to page limitations, we only give an informal description:

1) We forbid that empty clock zones have any connection or path variables established leading from or towards them.
2) We forbid that operations or wires are assigned to clock zones with an insufficient number of adjacencies for that very operation or wire.
3) We ensure that the number of established connections for each clock zone match the number of adjacencies of the assigned operation or wire in the netlist.

Overall, by the help of a reasoning engine we can use the presented symbolic formulation to determine whether a given Boolean function f represented in terms of a netlist can be realized on a QCA grid of size $N = x \times y$. Besides that, the symbolic formulation can be extended so that further objectives/constraints are considered. How this eventually can be applied in order to improve the current state-of-the-art in QCA design is illustrated by means of case studies in the next section.

IV. Evaluation and Application

This section discusses the applicability and the capabilities of the proposed design method. To this end, a tool has been implemented in C++ which automatically generates the symbolic formulation based on a given function f to be realized (represented in terms of a netlist) as well as the corresponding parameters such as grid size and objectives/constraints. As reasoning engine, we utilized the SMT solver Z3 [18] version 4.5.1 64 Bit. Based on that, we conducted evaluations and case studies which compare the respectively obtained QCA circuits against the state-of-the-art and demonstrate the capabilities of the resulting design method for the purpose of design exploration. All evaluations of our exact approach have been conducted on an Intel Xeon E5-2630 v3 machine with 2.40 GHz (up to 3.20 GHz boost) and 64 GB of main memory running Fedora 22. In the following, we summarize the evaluations and demonstrations.

TABLE I: Comparison to heuristic approach

	Heuristic [12]			Proposed approach		
Circuit	Area	CP	t in s	Area	CP	t in s
2:1 MUX	20×25	5	9	15×15	5	< 1
XOR	20×35	7	11	15×15	5	< 1
XNOR	30×30	8	13	20×20	8	2
Half adder	35×30	8	55	25×25	10	13
c17 [23]	50×30	13	15	25×30	16	56
ParGen [24]	45×50	14	27	35×30	14	791
ParCheck [24]	50×70	14	3014	20×60	16	1140
4:1 MUX	55×40	19	9612	35×35	22	5131

A. Comparison to Related Work

In order to demonstrate the quality of the solutions obtained by the proposed method compared to the state-of-the-art, we realized several benchmarks (given as Verilog netlists) which have recently been considered for evaluating the heuristic synthesis solution proposed in [12]. The resulting designs have been verified using the *Coherence Vector Simulation Engine* provided by the design tool *QCADesigner* [22]. Both approaches (the heuristic one from [12] as well as the exact one proposed in this work) aimed at minimizing the area of the QCA circuit, i. e. the size of the occupied grid of QCA cells. The respectively obtained results are reported in Table I, where column *Area* lists the resulting grid size, column *CP* the resulting critical path, and column *t in s* the required execution time in seconds to generate the result. Again, we are looking for the minimal *area* in this work. Thus, the critical path may be of arbitrary length.

The results first show that, indeed, determining results with minimal area as conducted in this evaluation is computationally expensive and requires a significant amount of time. However, as also can be seen, the proposed solution and the power of the applied SMT solver nevertheless allows for determining the desired results in reasonable time. Moreover, even though the heuristic approach has been executed on a machine with lower performance, the runtime is remarkably high.[2] For the first time ever, this allows to precisely evaluate previously proposed heuristic solutions. This unveils e. g. that the method proposed in [12] yields results that are almost three times larger than the actual minimum. That shows further potential for improving heuristic solutions to be investigated in future work. Besides that (and also in contrast to previous work), the proposed method additionally allows to enforce further objectives/constraints as demonstrated next.

B. Design Exploration

Existing design methods for QCA such as [9], [11], [12], [25] focus on single design objectives only and often do not consider further issues. In contrast, the symbolic formulation introduced in Section III can be extended so that designs result that satisfy further/other objectives/constraints. This allows for the *exact* exploration of different design configurations, giving

[2]The heuristic approach was executed on an Intel Core i7 4800MQ with 2.70 GHz (up to 3.70 GHz boost) and 8 GB of main memory running Windows 7.

TABLE II: Design exploration in terms of wire crossings

Circuit	Without crossings Area	CP	Single crossing Area	CP	With crossings Area	CP
2:1 MUX	15×15	5	15×15	5	15×15	5
XOR	15×15	5	15×15	5	15×15	5
XNOR	15×30	9	20×20	8	20×20	8
Half adder	25×30	10	25×30	8	25×25	10
c17 [23]	25×35	14	25×30	11	25×30	16
ParGen [24]	50×25	18	25×45	14	35×30	14
ParCheck [24]	20×70	17	35×35	12	20×60	16
4:1 MUX	40×35	16	25×55	16	35×35	22

TABLE III: Design exploration in terms of equal signal lengths

Circuit	Without crossings Area	CP	Area gain	Tp
2:1 MUX	15×15	5	0 %	$1/1$
XOR	15×15	5	0 %	$1/1$
XNOR	15×25	9	17 %	$1/2$
Half adder	25×30	13	0 %	$1/2$
c17 [23]	20×30	14	31 %	$1/3$
ParGen [24]	15×40	16	52 %	$1/4$
ParCheck [24]	20×50	13	29 %	$1/2$
4:1 MUX	30×40	24	14 %	$1/6$

the designer the possibility to identify the most adequate solution for his or her requirements. In the following, we demonstrate this for two representative design objectives.

First, we considered wire crossings, which we implemented via the multilayer approach discussed in [26].[3] In general, wire crossings in QCA designs are undesired as it requires the change between QCA layers or the application of rotated cells in case of coplanar crossings. Both affecting the reliability and performance [26], [3]. On the other side, allowing crossings could yield designs with much less area demands (or make a function realizable in the first place). Hence, designers might be interested whether and at what costs a function can be realized with a specific upper bound of wire crossings or even without any crossings.

Incorporating a limitation on the number of crossings can easily be implemented in the proposed solution by adjusting constraint C_2. Doing that allowing for no crossings, a single crossing or without any limitations on crossings eventually yields results as summarized in Tab. II. As can be seen, all considered functions can also be realized without crossings (assuming that I/1O placement is unrestricted). This causes an increase in the area by up to 20 % (*Half adder*), though. Based on these evaluations, a designer now can decide whether such an increase in area is worth avoiding crossings or not.

In a second evaluation, we considered the impact of constraining that all input signals of a logic cell arrive within the same clock cycle. Thus far, it is a common approach to design QCA circuits such that in each clock cycle new input data can be passed in leading to high throughput (Tp). This requires equal signal lengths, though, which is ensured by the proposed approach by means of constraint C_8. On the downside, additional wires might be necessary for synchronization purposes only. The proposed approach enables the exploration of the resulting trade-off between throughput and area.

More precisely, disabling constraint C_8 eventually yields results as summarized in Tab. III. Column *Area gain* lists thereby the area improvement and column Tp the resulting throughput, both with respect to the circuits listed in Tab. I (obtained with constraint C_8 enabled). The throughput reduction follows from the necessity to delay new input data in order to assure synchronicity amongst internal signals.

In case of the *ParGen* circuit, the area could be reduced by half – surely making it interesting to leave the comfort area at the costs of throughput reduction to $1/4$ of the maximum value. Comparing to that, the circuit *ParCheck* received a lower gain of 29 %. Nonetheless, it did so by only reducing its Tp by half. In case of the *4:1 MUX* circuit however, the throughput dropped to $1/6$ while only 14 % area gain could be achieved.

In a similar fashion, many further constraints/objectives could easily be considered and, hence, evaluated, using the proposed method.

[3] Since the formal model does not differ, the proposed method also supports coplanar wire crossings.

V. Conclusion

In this work, we presented an automatic and exact design method for QCA circuits. In contrast to existing (manual or heuristic) approaches, our method guarantees the satisfaction of desired design objectives and physical constraints. To this end, we proposed a symbolic formulation which captures whether a considered netlist can be realized on a QCA grid of size N adhering the given objectives/constraints. By leveraging formal reasoning engines for the symbolic formulation, we can determine QCA circuits satisfying the respective features. By this, it is possible to evaluate different design constraints/objectives in an *exact* fashion – providing significant potential for design exploration for QCA circuits. Experimental evaluations confirmed the benefits of the proposed approach. We were able to show that previously proposed design solutions (based on heuristics) generate results which are far from being optimal. Moreover, we demonstrated how the proposed method allows for the design exploration of QCA circuits with respect to different objectives/constraints.

References

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

[2] P. Singer, "Reframing the roadmap: ITRS 2.0," *Solid State Technology*, vol. 58, no. 1, p. 2, 2015.

[3] N. G. Anderson and S. Bhanja, *Field-coupled Nanocomputing: Paradigms, Progress, and Perspectives*, 1st ed. Springer, 2014.

[4] J. Timler and C. S. Lent, "Power gain and dissipation in quantum-dot cellular automata," *J. Appl. Phys.*, vol. 91, no. 2, pp. 823–831, 2002.

[5] J. Pitters, L. Livadaru, M. B. Haider, and R. A. Wolkow, "Tunnel coupled dangling bond structures on hydrogen terminated silicon surfaces," *JCP*, vol. 134, no. 6, 2011.

[6] S. Perri and P. Corsonello, "New methodology for the design of efficient binary addition circuits in QCA," *TNANO*, vol. 11, no. 6, pp. 1192–1200, 2012.

[7] E. Fazzion, O. L. Fonseca, J. A. M. Nacif, O. P. V. Neto, A. O. Fernandes, and D. S. Silva, "A quantum-dot cellular automata processor design," in *SBCCI*, 2014.

[8] M. Kianpour and R. Sabbaghi-Nadooshan, "A novel quantum-dot cellular automata CLB of FPGA," *J. Comput. Electron.*, vol. 13, no. 3, pp. 709–725, 2014.

[9] M. Bubna, S. Roy, N. Shenoy, and S. Mazumdar, "A layout-aware physical design method for constructing feasible QCA circuits," in *Great Lakes Symp. VLSI*, 2008.

[10] R. Ravichandran, S. K. Lim, and M. Niemier, "Automatic cell placement for quantum-dot cellular automata," *Integration*, vol. 38, no. 3, pp. 541–548, 2005.

[11] V. Vankamamidi, M. Ottavi, and F. Lombardi, "Two-dimensional schemes for clocking/timing of QCA circuits," *TCAD*, vol. 27, no. 1, pp. 34–44, 2008.

[12] A. Trindade, R. S. Ferreira, J. A. M. Nacif, D. Sales, and O. P. V. Neto, "A placement and routing algorithm for quantum-dot cellular automata," in *SBCCI*, 2016.

[13] E. P. Blair and C. S. Lent, "An architecture for molecular computing using quantum-dot cellular automata," in *IEEE-NANO*, 2003.

[14] 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.

[15] D. A. Reis, C. A. T. Campos, T. R. Soares, O. P. V. Neto, and F. S. Torres, "A methodology for standard cell design for QCA," in *ISCAS*, 2016.

[16] C. A. T. Campos, A. L. P. Marciano, O. P. V. Neto, and F. S. Torres, "USE: A universal, scalable, and efficient clocking scheme for QCA," *TCAD*, vol. 35, no. 3, pp. 513–517, 2016.

[17] J. Huang, M. Momenzadeh, L. Schiano, M. Ottavi, and F. Lombardi, "Tile-based QCA design using majority-like logic primitives," *JETC*, vol. 1, no. 3, pp. 163–185, 2005.

[18] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS/ETAPS*, Berlin, Heidelberg, 2008.

[19] D. Große, R. Wille, G. W. Dueck, and R. Drechsler, "Exact Multiple-Control Toffoli Network Synthesis With SAT Techniques," *TCAD*, vol. 28, no. 5, pp. 703–715, 2009.

[20] R. Wille, A. Lye, and R. Drechsler, "Exact reordering of circuit lines for nearest neighbor quantum architectures," *TCAD*, vol. 33, no. 12, pp. 1818–1831, 2014.

[21] O. Keszöcze, R. Wille, T. Ho, and R. Drechsler, "Exact one-pass synthesis of digital microfluidic biochips," in *DAC*, 2014, pp. 1–6.

[22] K. Walus and G. A. Jullien, "Design tools for an emerging SoC technology: Quantum-dot cellular automata," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1225–1244, 2006.

[23] V. S. Kalogeiton, D. P. Papadopoulos, O. Liolis, V. A. Mardiris, G. C. Sirakoulis, and I. G. Karafyllidis, "Programmable crossbar quantum-dot cellular automata circuits," *TCAD*, vol. 36, no. 8, p. 1, 2017.

[24] F. Ahmad and G. Bhat, "Novel code converters based on quantum-dot cellular automata (QCA)," *IJSR*, vol. 3, no. 5, pp. 364–371, 2014.

[25] T. Teodosio and L. Sousa, "QCA-LG: A tool for the automatic layout generation of QCA combinational circuits," in *Norchip*, 2007, pp. 1–5.

[26] B. IL and P. P., "Two-layer synchronized ternary quantum-dot cellular automata wire crossings," *Nanoscale Res. Lett.*, vol. 7, no. 1, 2012.