DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Electrical Engineering

Signal Distribution Networks in Automatic QCA Standard-Cell Placement and Routing

Benjamin Hien

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Electrical Engineering

Signal Distribution Networks in Automatic QCA Standard-Cell Placement and Routing

Signalverteilungsnetze in Automatischer QCA Standarzellen Plazierung und Verdrahtung

Author:	Benjamin Hien
Supervisor:	Prof. Dr. Robert Wille
Advisor:	Dr. Marcel Walter
Submission Date:	26.01.2023

I confirm that this master’s thesis in electrical engineering is my own work and I have documented all sources and material used.

Munich, 26.01.2023 Benjamin Hien

Abstract

In this thesis, a new approach for placement and routing in Quantum-dot Cellular Automata (QCA) is presented by introducing Distribution Networks (DNs). These networks enhance the functionalities of an already existing scalable placement and routing algorithm. The first DN, called Ordering Distribution Network (ODN), reduces the number of wire crossings by an average of 13% and the layout area by an average of 17% by reorganizing primary inputs. The second DN, called Majority Gates Distribution Network (MGDN), enables the algorithm to place and route "+"-majority gates while satisfying timing constraints, resulting in an average of 865% increase in layout area despite the fact that in average 72% fewer gates are required for logic networks containing majority gates. Notably, the MGDN does not incorporate any wire crossings, thus reducing the number of wire crossings by and average of 40% in the layout. Lastly, the third DN, called Sequential Distribution Network (SDN), is the first to enable the placement and routing of sequential logic in QCA. The proposed method is evaluated through extensive simulation and experimentation and shows different trade-offs in terms of design metrics like performance, layout area or number of wire crossings.

Contents

1 Introduction

Quantum-dot Cellular Automata (QCA) is a promising technology for the design of ultra-low power, high-density, and high-performance digital circuits [44]. QCA technology is based on the manipulation of the electrostatic states of quantum dots (QDs) to perform logic operations. The potential benefits of QCA include its ability to operate at extremely low power levels and high device density [55].

1.1 Motivation

Although the theory sounds very promising, the design of QCA circuits is a challenging task due to the complexity of the underlying physics and the lack of appropriate design tools [49]. Placement and routing are two critical steps in the design of QCA circuits that determine the overall performance, power consumption, and area efficiency of the circuit.

Placement refers to the process of arranging the QCA cells on the chip. It involves determining the optimal position of the cells to minimize the number of interconnects and the routing area [28]. Routing, on the other hand, refers to the process of connecting the QCA cells to form a functional circuit. It involves determining the optimal path for the interconnects to minimize the routing area [28]. In QCA placement and routing are strongly related to each other and have to be viewed as a connected process. The placement and routing is a non-trivial task due to the constraints imposed by the QCA technology, such as local and global synchronization constraints [71].

The importance of placement and routing in the design of QCA circuits cannot be overstated. The performance, power consumption, and area efficiency of the circuit are all directly affected by the placement and routing of the QCA cells. An optimal placement and routing can significantly improve the performance and reduce the power consumption of the circuit, making it more suitable for practical applications [69].

In this thesis, we will explore the various placement and routing techniques used in the design of QCA circuits. We will emphazise the need for scalable placement and routing algorithms and investigate the trade-offs between performance, costs, and area efficiency, and we will propose new techniques to improve the design of QCA circuits.

1.2 Objective

Hence, the limitations of the current state of the art scalable placement and routing algorithms should be overcome by introducing so called *Distribution Networks* (DNs). The proposed DNs add new functionalities to a scalable placement and routing algorithm, such as the ability to reduce wire crossings and layout area, place and route majority gates and sequential logic.

The first DN, called the *Ordering Distribution Network* (ODN), is designed to reduce wire crossings and layout area by introducing a new ordering of primary inputs (PIs). This improves the overall area required for the circuit and reduces the number of wire crossings and therefore the costs of the designed QCA circuits.

The second DN, called the *Majority Gates Distribution Network* (MGDN), is designed to enable the placement of majority gates in the QCA circuit. However, it is important to note that while the number of gates placed is reduced, the layout area may actually increase due to the timing constraints.

The third DN, called the *Sequential Distribution Network* (SDN), enables the algorithm for the first time to place and route sequential logic in QCA. This is a significant advancement as current placement and routing algorithms are not able to handle sequential logic.

The main objective of this thesis is to improve the performance and costs of the scalable placement and routing algorithm for QCA circuits by introducing these new DNs. The proposed DNs have the potential to improve the performance and costs of the algorithm while enabling the placement and routing of majority gates and sequential logic, which were previously not possible. This thesis aims to provide a deeper understanding of the challenges and opportunities of QCA technology and to contribute to the development of practical design tools for QCA circuits.

2 Preliminaries

This chapter establishes a theoretical basis consisting of declarations and definitions required for the understanding of the ideas and their implementations proposed in this work. The four fields forming this basis are the representation of logic circuits (Section 2.1), QCA technology (Section 2.2), the placement and routing problem (Section 2.3) and sequentiality (Section 2.4).

2.1 Representation of logic circuits

Logic circuits provide a powerful construct that allows an abstraction of digital circuits to a logic level and thereby makes it possible to discuss and argue about them scientifically. This abstraction was made possible by the Boolean algebra, formed by the mathematician George Boole in 1847. It shows that every digital circuit can be represented by logic functions, independent of their underlying technology [22]. In the following sections first, a definition of a Boolean algebra is given, and second, it is shown how logic networks can be formed using them.

2.1.1 Boolean Functions

A definition of Boolean calculus was first provided by Edward V. Huntington in 1993. From *set of independent postulates for the algebra of logic* and his own correction [25, 24], the following equations form the basis of every Boolean algebra:

Definition 2.1.1 (Basis for Boolean algebra)**.** Let a, b, c be arbitrary elements of an abstract algebra $(L, +, ')$ with their set denoted as B_{abc}. The algebra includes the binary function disjunction $+ : B_{abc} \times B_{abc} \to B_{abc}$ and the unary function $' : B_{abc} \to B_{abc}$.

$$\begin{aligned} a + b &= b + a \\ (a + b) + c &= a + (b + c) \\ (a' + b')' + (a' + b)' &= a \end{aligned}$$

The last of his postulates is named after the inventor and is commonly known as *Huntington equation*.

There also exists an "universe element" $u \in B_{abc}$ for which holds:

$$\exists u' : a + u' = a$$
$$\exists u : a + a' = u.$$

Despite the capability of the disjunction and negation operators to construct a Boolean algebra, it is common practice to additionally utilize the conjunction operator $\cdot : B_{abc} \times B_{abc} \to B_{abc}$ to create more concise and comprehensible logical expressions. The most widely used Boolean algebra, denoted as $\mathbb{B}$, is defined as the tuple $(B_{abc}, \vee, \wedge, \neg)$, where $\vee$ and $\wedge$ are alternate notations for the binary disjunction operator $+$ and conjunction operator $\cdot$ within $\mathbb{B}$. The third function, $\neg$, represents the unary negation operation, previously denoted as $'$. The set B_{abc} comprises of two distinct elements, namely $0, 1$, with $u = 1$ and $\neg u = 0$ in accordance with the definition in [21].

Given that this definition is limited to the utilization of only the three Boolean functions $(\vee, \wedge, \neg)$, an extension to this definition is proposed in [53].

Definition 2.1.2 (Boolean function)**.** A Boolean function can be described as $f : \{0,1\}^k \to \{0,1\}$, with $k \in \mathbb{N}^*$ being the number of arguments or arity of the function. A function with k arguments is referred to as k-ary. Multi-output Boolean functions can be described as $\{0,1\}^k \to \{0,1\}^m$, with $k \in \mathbb{N}^*$ and the integer $m > 0$.

Nonetheless, every k-ary function can still be decomposed into a set of common Boolean functions $(\vee, \wedge, \neg)$. One example for this is the 3-ary majority function, which is a very important Boolean function for QCA technology.

Definition 2.1.3 (Majority Function)**.** The ternary Boolean majority function is defined as: $\langle a, b, c \rangle = ab + ac + bc$, so that the function value equals the majority of it's incoming values.

It follows: $\langle a, b, 0 \rangle = a \cdot b$ and $\langle a, b, 1 \rangle = a + b$.

Common notations for Boolean functions are *conjunctive normal form* (CNF) or *disjunctive normal form* (DNF), using literals.

Definition 2.1.4 (Literal)**.** A literal is either an atom a (positive literal) or the negation of an atom $\neg a$ (negative literal).

Definition 2.1.5 (CNF and DNF)**.** A propositional Boolean formula is said to be in CNF if it is a conjunction of *clauses*, each of which is a disjunction of literals [74]:

$$\bigwedge_i \bigvee_j (\neg) v_{ij},$$

where $v_{ij} \in \mathbb{B}$.

A propositional Boolean formula is said to be in DNF if it is a disjunction of clauses, each of which is a conjunction of literals:

$$\bigvee_i \bigwedge_j (\neg) v_{ij},$$

where $v_{ij} \in \mathbb{B}$.

Using the CNF or rather the DNF, Definition 2.1.1 and *De Morgan's laws* [26]:

Definition 2.1.6 (De Morgan's laws)**.** Given a Boolean Algebra $\mathbb{B} = (B_{ab}, \vee, \wedge, \neg)$ with two arbitrary elements $a, b \in B_{ab} = 0, 1$, the following logic principles can be applied:

$$\neg a \wedge \neg b = \neg(b \vee a)$$
$$\neg a \vee \neg b = \neg(b \wedge a),$$

it follows that any Boolean algebra can be reduced to only two operands, e.g., conjunction ($\vee$) and negation ($\neg$) or disjunction ($\wedge$) and negation ($\neg$). Any set of such two Boolean functions is called *universal*.

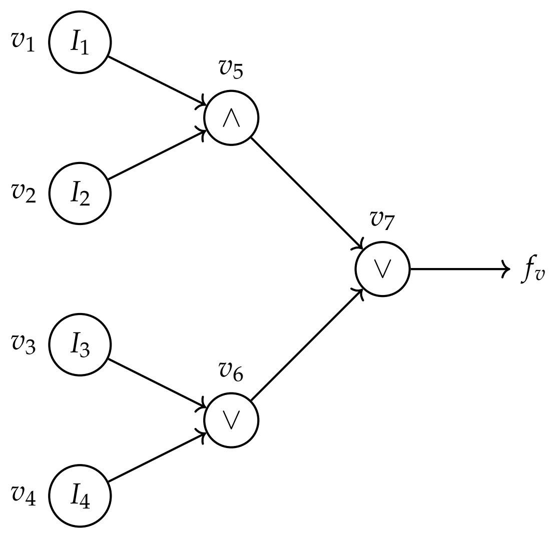
2.1.2 Logic Networks

There are many ways of representing Boolean Functions. But all *canonical* forms like truth tables (TTs), reduced sum of products (RSOPs) or binary decision diagrams (BDDs), suffer from exponential representations, making them impractical for big logic circuits. Even if a reasonable representation exists for a given function, simple operations like forming the complementary could yield an exponential function representation [45].

Logic networks overcome these restrictions by being *non-canonical*, meaning that a given function can be represented by different logic networks. The following definition for Logic Networks is derived from [11]:

Definition 2.1.7 (Logic Network)**.** A logic network $N(V, E)$ is a rooted, directed graph with vertex set V and edge set E. For any vertex $v \in V$, vertices connected by incoming edges $e_{inc} \in E$ are called parent. A vertex connected by an outgoing edge $e_{out} \in E$ is called child. V contains two types of vertices. A *non-terminal* vertex v has as attributes an argument index $index(v) \in \{1, ..., n\}$, and l parents $parent_1(v), ..., parent_l(v) \in V$. A *terminal* vertex v has as attribute a value $value(v) \in \{0, 1\}$.

Furthermore, for any non-terminal vertex v, if $parent_i(v)$ with $1 \leq i \leq l$, then we must have $index(parent_i(v)) < index(parent_l(v))$ respectively.



The corresponding recursive Boolean function reads:

$$\begin{array}{ll} f_v = f_v(v_7) & f_v(v_6) = f_v(v_3) \wedge f_v(v_4) \\ f_v(v_7) = f_v(v_5) \vee f_v(v_6) & f_v(v_5) = f_v(v_1) \vee f_v(v_2) \end{array}$$

with primary inputs: $f_v(v_1), f_v(v_2), f_v(v_3), f_v(v_4) \in \{0, 1\}$

Figure 2.1: Binary Logic Network

This definition allows vertices to have an unrestricted number of parents, implying that the Boolean function represented by the vertex can be k-ary:

Definition 2.1.8 (Logic Network Boolean Functions)**.** A set of k-ary Boolean Functions $x_1, ..., x_n \in \mathbb{B}$ is assigned to every vertex via the argument index $index(v) = i$. The graph function f_v is defined recursively as:

1. If v is a terminal vertex:
    a) If $value(v) = 1$, then $f_v = 1$
    b) If $value(v) = 0$, then $f_v = 0$
2. If v is a non-terminal vertex with $index(v) = i$, then f_v is the function:
   $f_v(v_i) = x_i(f_{parent_1(v)}(v_{i-1}), ..., f_{parent_l(v)}(v_{i-n}))$.

The recursive nature of the Boolean Function definition in logic networks can be seen in Figure 2.1.

The non-canonical property can be explained by the fact that nodes with the identity function are allowed, which can be inserted everywhere in the logic network, while the function representation of the logic network stays the same. Even the exclusion of such identity nodes has no impact, since simple node combinations, like two negation nodes, collapse to the identity function. Following this argumentation, there exists an infinite number of logic networks representing each one Boolean Function, resulting in the widely accepted assumption, that the determination of an optimal logic network is an $\mathcal{NP}$-complete problem [67]. Attempts to create canonical logic networks, seem to evade this problem, but include *co*$\mathcal{NP}$-complete problems in itself [11]. Nevertheless, logic networks have proven to be very useful in transforming logic circuits into gate representations. This process is called *logic synthesis*. Due to the complexity of the representations algorithms commonly used for logic synthesis are based on approximate solutions.

Adapting from the definitions in [67], a terminal vertex is referred to as *primary input* (PI) with their set denoted as I. The set of non-terminal vertices referred to as *nodes* is denoted as Λ. The definition requires $I \cap \Lambda = \emptyset$. An edge connecting a parent v_i and a child vertex v_j is called a *signal*. With $i < j$ the notation of a signal is given as (v_i, v_j). The set of all signals is denoted as Σ. If an edge is dangling, so it doesn't point to another vertex, it is called *primary output* (PO) and their set is denoted as O. Therefore also $\Sigma \cap O = \emptyset$ holds. From the definition of a logic network, we can now describe it as acyclic directed graph $N = (\Lambda, I, \Sigma, O)$.

As already mentioned in Section 2.1.1, a universal set of two Boolean functions can form any Boolean algebra. As long as this universality is contained, the set of node functions in a logic network can be extended arbitrarily. Common logic networks containing only two network functions are e.g. *AND-Inverter Graphs* (AIGs) [5] allowing only conjunction and negation. Another binary logic network, which is used in the QCA domain, is the *Majority-Inverter Graphs* (MIGs) [5] utilizing the ternary majority function and negation. But there also exists a wide range of logic networks that permit more than just two-node functions. One example is the *AND-OR-Inverter Graph* (AOIG) with conjunction, disjunction, and negation functions, respectively [67].

As part of the logic synthesis, a suitable logic network representation of the combinational circuit has to be determined. Because, even though these logic networks can implement any Boolean function given by a specification, not every logic network can be synthesized into any given technology. Looking at the current standard technology *complementary metal-oxidesemiconductor* (CMOS), the logic network is then synthesized by using building blocks consisting of *metal-oxide-semiconductor field-effect transistors* (MOSFETs), the elemental unit in this technology.

2.2 QCA Technology

In order to fulfill the well-known Moore's law [51], demanding a doubling of transistors on a chip every two years, CMOS technology is facing a multitude of challenges. Most notable are the short-channel effect, impurity variations, and most importantly, the heat dissipation resulting from static and dynamic power losses [35, 63, 73]. To tackle these challenges among others the International Roadmap for Devices and Systems (IRDS, former by ITRS), provides a platform proposing solutions within the semiconductor domain, e.g. new materials and multi-core architectures. Also new technologies are being researched, including quantum computing and the domain of *Field-Coupled Nanocomputing* (FCN) [49]. This work focuses on one of the most promising FCN technologies, namely *Quantum-dot Cellular Automata* (QCA). The main difference of this technology compared to CMOS is the representation of logical modes, using the location of electron pairs in QCA-cells, rather than voltage levels. Data between cells are transferred based on Coulomb repulsion, using electromagnetic fields, enabling the technology to achieve high performance in terms of device density, clock frequency, and power consumption [40]. Hence, QCA tackles exactly the main issues faced by CMOS technology and provides a promising digital system for the future [2]. However, QCA also presents its own challenges in terms of the manufacturing process, manufacturing standards, and different design methodologies [39]. Because this work focuses on the design of QCA circuits, it mainly views circuit parameters such as area, performance, and wire crossings [2]. In order to allow an analysis under these parameters, first the QCA technology and the resulting design constraints have to be understood. Therefore, Section 2.2.1 introduces QCA cells as building blocks for QCA gates, which are discussed in Section 2.2.2. After understanding the clocking in Section 2.2.3, the constraints for the placement and routing problem can be formulated subsequently. In addition sequential behavior is discussed in CMOS and transferred to the QCA domain in Section 2.4.

2.2.1 Cells

As already mentioned, in QCA technology logical states are no longer represented by voltage levels but by the location of electrons [6]. In order to achieve this property a nanosized structure is needed, which is capable of trapping electrons in a certain position. The utilization of so-called *quantum-dots* (QDs) serves as the foundation for this study. QDs can be implemented in various ways, including the use of several to one hundred atoms of a semiconductor in CMOS-QCA technology [52], the implementation of nanomagnets in mQCA technology [42], or the use of patterned dangling bonds on hydrogen passivated silicon referred to as SiDBs [48]. As a result, quantum mechanics

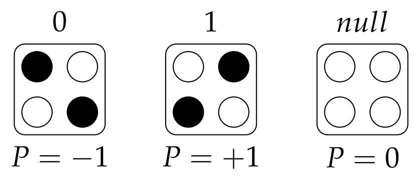


Figure 2.2: QCA-Cell sates

applies to the electrical properties of QDs. For this work it is sufficient to understand that every QD has a bound state, where a particle tends to be localized and the bound state is subject to a potential, which can be external or due to the presence of other particles. Because this enables QDs to have discrete electronic states, they are also referred to as *artificial atoms*. A combination of them is used to build a QCA cell, also known as *artificial molecule* [52]. Figure 2.2 shows three such QCA cells, where the four circles at the corners of each QCA cell represent the QDs or rather their quantum barriers, which are capable of trapping each one electron. In addition, a cell contains two excess electrons, which can be localized by quantum dots. When a QD currently traps an electron, it is depicted black and an unoccupied QD is depicted white. Coulomb repulsion causes electrons to occupy diagonally opposed QDs, resulting in three possible cell states [50, 33, 34].

A stable state indicates that it is easily distinguishable from the usual energy band. Therefore, the energy difference between two consecutive energy states must be well above the thermal noise energy ($k_B T$). Only such states are suited for information transfer. The stable states can be derived from cell polarization, which can be $+1$ and -1 or *null* in the unexcited state. The two stable states contain the same electrostatic energy and are used to encode the binary values 0 and 1 [50]. Figure 2.2 shows three cells with possible states and their resulting polarizations and logical states.

As already stated, QDs can be influenced externally, allowing the designer to fix the polarization of a QCA cell. This effect is used to input information into one QCA cell, called the driver cell. When a driver cell is placed side-by-side with other QCA cells, its polarization causes the polarization of the adjacent cell to change. When the adjacent cell is polarized, it passes its state again to the next cell and so on [33]. Figure 2.3 depicts such a structure, where the left-most cell represents the driver cell with fixed polarization representing a logic "1". With every cell passing its polarization to the cell to its right, the polarization of the right-most cell can be measured and the logic value, which propagated through the structure can be extracted. Due to the property of just passing the information from its input to its output, the shown structure represents a QCA wire segment.

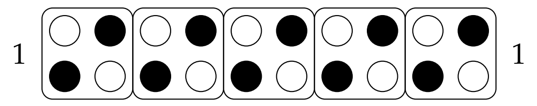
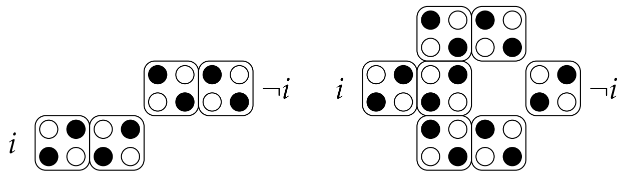


Figure 2.3: Adjacent QCA-cells forming a wire segment



(a) Diagonal Inverter (b) Standard Inverter

Figure 2.4: Different QCA Inverter representations

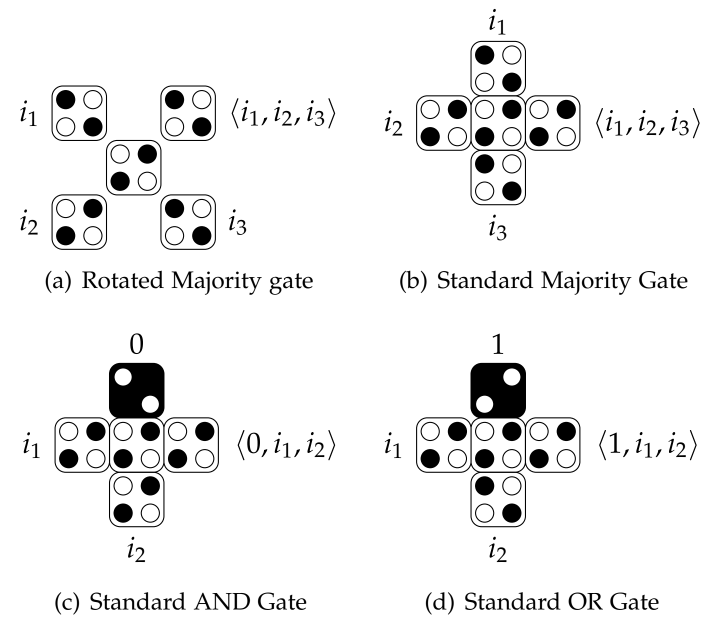
2.2.2 Gates

In this subsection, QCA cells are combined to form different logic gates, which form the gate library used later for the design of QCA logical circuits. Some of these gates are inherited from the QCA ONE library [46], which is already used fully [43] or partially [71, 17] as a basis for some existing works.

The QCA ONE library proposes gates formed by one tile as well as gates formed by multiple tiles. A tile describes a uniform size of gates. For the QCA ONE library a tile is of dimension 5×5 cells. A major drawback of this library is the use of some gates that are highly prone to crosstalk and the implementation of sequential circuits that do not align with the theory of this work [46]. Therefore, the gate library used in this work is presented below. The sequential theory is provided in Section 2.4.

Inverters and majority gates are the main building blocks of QCA circuits. Starting with the inverter or NOT gate, the simplest implementation is shown in Figure 2.4(a). It consists of two wire segments which are shifted by exactly one cell height, so that the polarization is transferred diagonally resulting in an inversion of the input [50]. In order to get a more robust gate regarding disturbance, the C-shaped inverter shown in Figure 2.4(b) was introduced [46]. This gate is used as standard in many libraries and works [43, 17, 36, 13], but it should be mentioned that this implementation is really prone to complex single-electron faults [37] and even common displacement faults [54], suggesting the addition of an inverter leg that results in an E-shaped gate structure. Nevertheless the C-shaped inverter gate is part of the QCA ONE library and is also selected as standard gate for this work.

Due to its importance in QCA technology, the next gate, which needs to be investigated, is the majority gate. Definition 2.1.3 suggests that the implementation of this


i_1
$\langle i_1, i_2, i_3 \rangle$
i_2
i_3
i_1
i_2
$\langle i_1, i_2, i_3 \rangle$
i_3
0
i_1
$\langle 0, i_1, i_2 \rangle$
i_2
1
i_1
$\langle 1, i_1, i_2 \rangle$
i_2

(a) Rotated Majority gate (b) Standard Majority Gate

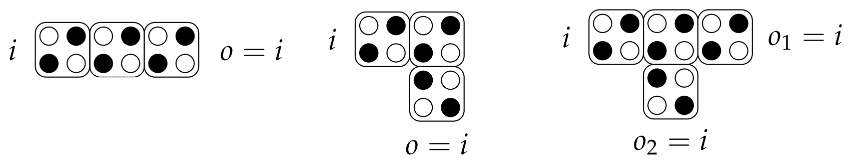
(c) Standard AND Gate (d) Standard OR Gate

Figure 2.5: The QCA Majority gate

function in CMOS technology requires multiple AND and OR gates to be placed and routed. In QCA technology on the other hand a majority function can be represented by exactly one gate, making it one of the major advantages over CMOS technology. There are two main implementations of the majority gate. The rotated majority gate in Figure 2.5(a), which is used in QCA ONE, and the $+$-majority gate shown in Figure 2.5(b). Both of these implementations have their advantages and drawbacks. On the one hand, the rotated majority gate exhibits a sufficiently high degree of fault tolerance against cell displacement or misalignment but has a very poor degree of fault tolerance against single cell omission or extra cell deposition [30]. On the other hand, the $+$-majority gate is very prone to cell displacement, but it is also used as a building block for the AND and OR gates in most works. This means that the fabrication process for all these gates is very similar and since this work is aimed to enhance the design process of QCA circuits, the $+$-majority gate is chosen as standard gate for this work.

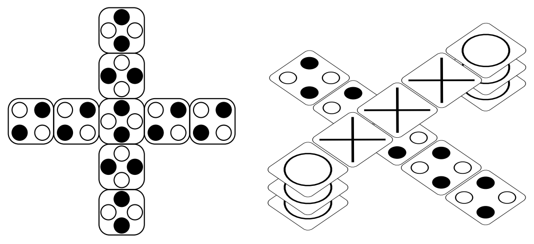
Following Definition 2.1.3 the AND gate can be derived by fixing one input of the majority gate to logic 0, while the OR gate is obtained by fixing one input to logic 1. The resulting gates, which are also part of the standard gate library of this work are shown in Figure 2.5(c) and Figure 2.5(d).

As previously demonstrated in Figure 2.3, a QCA wire also comprises of QCA cells and thus, unlike in CMOS technology, in QCA, wires must also be treated as gates.



(a) Straight wire (b) Bent wire (c) Fan-out wire

Figure 2.6: QCA wires

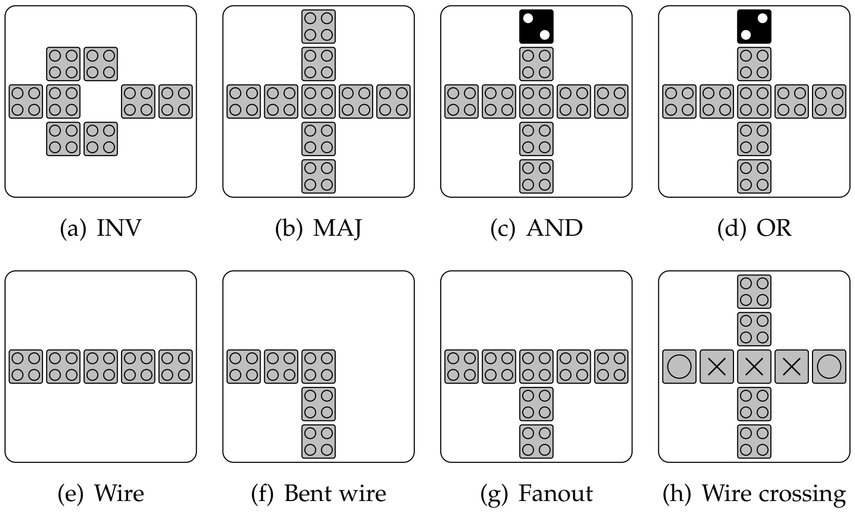


(a) Coplanar wire crossing (b) Multi-layer wire crossing

Figure 2.7: Different QCA wire crossing implementations

Since wires do not add any additional functionality to the logic, they are considered to represent the identity function. This property has the significant drawback of the cost of wires being comparable to that of other logic gates, which is often used as a major cost metric in circuit design. Until now, only straight planar wires have been considered. From the implementation of the majority gate, it is already evident that data is not only transferred in the x-dimension but also in the y-dimension, requiring the wiring to also be able to flow in both dimensions. When gates are placed side-by-side in two dimensions, the resulting circuit can be viewed as a 2D-grid. To allow information to change its propagation direction from the x-direction to the y-direction and vice versa, bent wires must also be introduced. They are depicted in Figure 2.6(b) and show a 90-degree bend. Since all tiles can be rotated by 90°, 180° and 270° respectively, a tile connected to a bent wire can be routed to each adjacent tile of a bent wire. Additionally, since all gates introduced so far have a fan-out of one, we need a fan-out node to duplicate signals. This is achieved by adding a bent wire to a straight wire, resulting in the fan-out shown in Figure 2.6(c).

The last special case of wires is the crossing case. By rotating the cells of one wire string by 45°, the rotated cells do not couple with non-rotated cells [54], as shown in Figure 2.7(a). This solution is very useful because it supports the planar structure of the circuit and is therefore referred to as *coplanar wire crossing*. Additionally, the


(a) INV (b) MAJ (c) AND (d) OR

(e) Wire (f) Bent wire (g) Fanout (h) Wire crossing

Figure 2.8: QCA Standard Library

possibility of multi-layer QCA has been investigated [19] and found to be particularly useful in the case of wire crossings. To achieve this, one wire string is raised to an additional higher layer, which is connected with a vertical interconnect, as shown in Figure 2.7(b). The signal transmission in the vertically-stacked cells works in the same way as in the horizontal direction. To prevent any crosstalk between the wire strings, intermediate layers of cells are used in the vertical direction. Theoretically, the top layer can not only be used as a wire, but since the signal distribution works in the same way as in the ground layer, gates can also be placed in these multi-layers. Simulations have shown that coplanar crossovers significantly reduce the coupling between the horizontal wire segments, making the horizontal interconnect very sensitive to crosstalk and therefore highly prone to cell displacements [54]. Multilayer circuits, on the other hand, show high robustness and are therefore used as a standard in this library. In the tile representation of the multilayer interconnect, the top wire string is represented with a $\times$, while the vertical layers are represented with a circle. It should be noted that the complex structure of the multilayer wire crossing yields high costs and is therefore avoided in the design of QCA circuits. In Figure 2.8, all gates used in this work are summarized.

2.2.3 Clocking

As mentioned above, data transfer in the QCA paradigm is accomplished by cell-to-cell interaction. Given a fixed polarization of a cell, the next cell reacts to the Coulomb

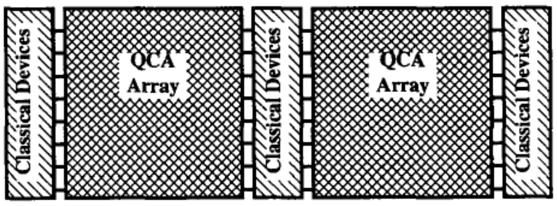


Figure 2.9: Schematic of a combined QCA and CMOS system [34].

repulsion and changes its polarization accordingly. Looking back at the wire segment in Figure 2.3, the leftmost cell has a fixed polarization and is called the input. After some time the information propagates through to the rightmost cell, representing the output of the simplified QCA-circuit. Generally, a QCA circuit can be seen as an assembly of cells on a two-dimensional grid or array, where each cell has a position with x and y coordinates assigned. Every cell with fixed polarization is called an input cell and drives the other cells gradually into matching polarization. When all cells have matching polarization, meaning that the electrons in two adjacent cells have the maximum distance and therefore minimum energy following Coulomb repulsion, the QCA array is said to be in *ground state* [33]. When a cell does not further contribute to the signal propagation of the circuit, it is called an output cell. While the polarization is propagating through the array, the direction of the propagation is always pointing away from the input cells and to the output cells. In reality the propagation doesn't go gradually through the array but shows a quite unpredictable behavior [34]. This is the first reason, why a *clocking*, involving well-defined states for the polarization of the cells and enabling a well-ordered signal propagation, was introduced. Another reason for a clocking is, that for the described straight forward process called *abrupt switching* [33] with dissipative coupling to the environment, the QCA-array has to be embedded into a CMOS environment, as shown in Figure 2.9. In the following, a short evolution of this primitive clocking to the currently used clocking is described.

Therefore, the QCA-Array is divided into smaller decoupled sub-regions called *clock zones* and each clock zone receives an external signal, a clock, assigned [33]. The clock can then activate and deactivate the cells of a zone in a way that the information propagates gradually from one zone to the next through the whole QCA circuit. In the approach first used, the clock decreases the QD barriers of all cells in a clock zone, when applying a new input. This means that the electrons are not trapped and can move freely following Coulomb repulsion, therefore taking over the polarization of the input cell. When all cells in the region are stable, the barriers are raised again, localizing the electrons in the cells, which now have the desired polarization. Meanwhile, the barriers of the subsequent clock zone are lowered simultaneously, the previous sub-region acts

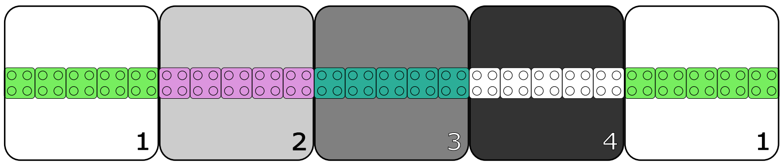
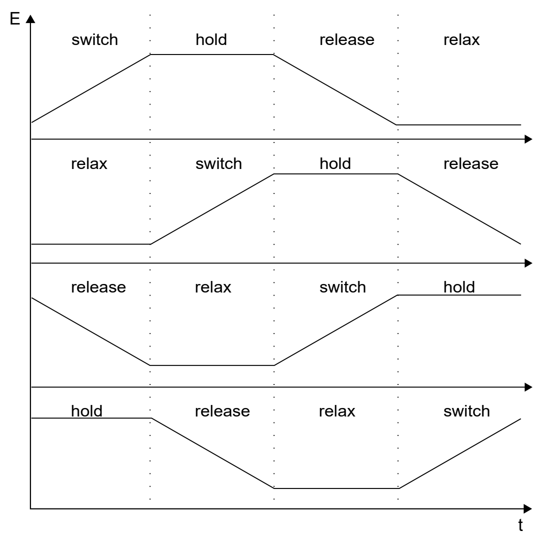


Figure 2.10: QCA wire divided into the four clock zones according to Bennett clocking

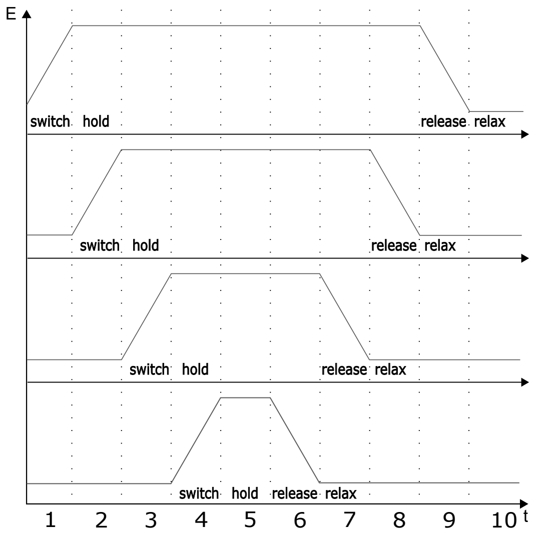
as fixed input and again the polarization is taken over. In this way, the information gradually propagates through the whole circuit [33]. Today's used approaches create electrical fields with an external clock generator and distribute it to the cells through the device substrate using embedded electrodes. Thereby, the energy level of the *null* state can be controlled, resulting in an equivalent effect as in the former approach [67].

A wire divided into clock zones is shown in Figure 2.10. The colors of the zones and cells, as well as the zone number, provide redundant information about the type of clock zone. They differ in the external applied electrical field and, therefore, the energy of the cells. In QCA, the clocking is divided into four consecutive states: *switch*, *hold*, *release*, and *relax*. They are aligned in a pipeline-like structure, where each of these states is phase-shifted by $\frac{\pi}{2}$, forming a 2π clock cycle. In the switch phase, cells start to get polarized, depending on the polarization of the driving cell. When the cells are polarized, they get fixed in the hold-phase. Afterwards, in the release-phase, the excitation gradually decreases, resulting in the unexcited relax-state [50]. After one clocking cycle, the next clocking cycle starts with the same order of states, also denoted with numbers $i = 1, 2, 3, 4$. Hence, for consecutive clocking numbers $(i_{next} = i_{previous} + 1 \mod clk)$ holds. The scheme of such a pipeline as clocking is depicted exemplary in Figure 2.11(a).

The described clocking is known as *Landauer clocking* [33]. Its name giver, Rolf Landauer, pointed out the significant power dissipation of this clocking mechanism. The main cause of high power dissipation is the *erase* function, which occurs because in Landauer clocking the release state directly follows the hold phase, irreversibly erasing the information and, therefore, transforming it into heat [31]. To address this issue in the QCA domain, Landauer proposed that the erase function must be eliminated from the clocking. He argued that every erased bit dissipates at least $k_B T \ln(2)$ in heat dissipation [29]. For example, if a QCA-cell has a size of $1nm \times 1nm$ and an operating frequency of $100GHz$, the corresponding density of devices results in 10^{14} cm^{-2}. Additionally, if a dissipation of $0.1eV$ every clock cycle is assumed, it results in a total power dissipation of $160kWcm^{-2}$. This directly implies that a device operating with this clocking would be inoperable (it would evaporate due to heat) [32]. The *Bennett clocking* [32] tackles exactly this problem by altering the timing of the clocking



(a) Landauer clocking


E
switch hold release relax
switch hold release relax
switch hold release relax
switch hold release relax
1 2 3 4 5 6 7 8 9 10 t

(b) Bennet clocking

Figure 2.11: QCA clocking pipeline

signals. Just as in the Landauer clocking, the clocking wave propagates in one direction, but leaving no trailing edge, when information is passed. Instead, the cells will be held in the excited state until the information propagates through the whole QCA-array. When the output is read, the excitation is released in reverse order resulting in no erase functions. This means, that this *quasi-adiabatic* clocking leads to a minimal power dissipation but with two constraints: (1) The effective clock rate is at least halved due to the additional backwards propagation and (2) since only one signal vector can be transmitted through the system, the pipeline capabilities are reduced [32]. The resulting clocking scheme for Bennett clocking is shown in Figure 2.11(b). In order to preserve the pipeline like behavior of the Landauer clocking and the energetic advantages of the Bennett clocking, QCA circuits are divided into sub-regions, which are each Bennett clocked.

As already stated, the QCA-array has to be divided into clock zones, in order to apply a proper clocking. Allowing an arbitrary number of cells in one clock zone gives lots of freedom in designing clock zones with variable geometries. This clocking is referred to as *cell-based* and increases the fabrication process due to its variety in clock zone geometries. Assuming the necessity of a uniform fabrication in order to fabricate circuits with millions of cells, this clocking gets infeasible for large circuits. Since in this scheme single cells can be clocked, also electrodes of the same size must be fabricated,

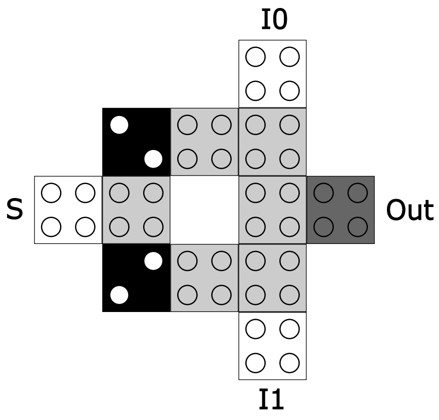
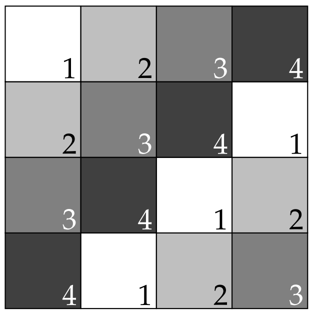
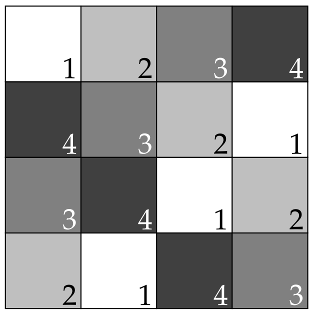


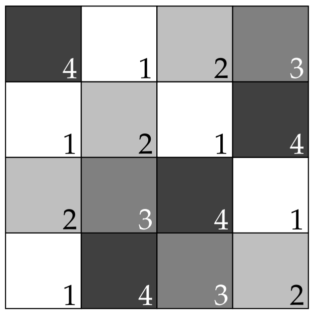
Figure 2.12: Cell based layout of a 2:1 MUX [38]



(a) 2DDWave [66]



(b) USE [13]



(c) RES [20]

Figure 2.13: Different clocking Schemes in QCA

in order to provide a clocking signal to the single-cell region. Since this is also not feasible, this design is obsolete [9]. An example of a cell-based clocking design of a 2:1 multiplexer (MUX) can be seen in Figure 2.12. To achieve uniform clock zones with a possible distribution of clocking signals, the *tile-based* clocking is introduced. The approach of this design is to provide uniform tiles of size 3×3 or 5×5. For clocking tiles larger than this, information propagation was suggested to be erroneous, also being an argument against cell-based clocking [58].

The tile-based clocking leads to several proposals for clocking schemes, which provide a specific distribution of clock zones. Since they follow a uniform pattern, they can be easily extended for any size of the circuit. In Figure 2.13, three clocking schemes are shown, each based on a different idea. Since information flow is only allowed in ascending clock order modulo *clk*, the 2DDWave clocking scheme in Figure 2.13(a) only allows information to propagate in two directions, south and east [66]. This simplicity prohibits back propagation, limiting the placement and routing of sequential circuits. It also restricts gates in the scheme to have a maximum input size of two, thereby not allowing the placement and routing of majority gates. The USE scheme, shown in Figure 2.13(b), addresses the first problem by introducing clocking loops into the scheme, providing the possibility to place sequential circuits [13]. The RES scheme, shown in Figure 2.13(c), addresses the second problem by giving the opportunity to place gates of input size three. Since one tile can connect to only four adjacent tiles, one of which must output the information of the gate on the tile, this gives the maximum input size allowed. This is especially important for the placement of majority gates [20]. In QCA technology, they can be represented by only one tile, making them one of the key advantages of QCA technology.

The basic building blocks of QCA have been identified and their method of data propagation through clocking has been established. With this understanding, the design and construction of QCA circuits can be undertaken. The critical aspect of this process is the proper placement and routing of these building blocks. In the following section, the placement and routing problem in QCA circuit design will be defined.

2.3 Placement and routing problem

In this section, the placement and routing (P&R) problem is formally defined using the theoretical foundations of gates and clocking established in the preceding sections. In simple terms, the (P&R) problem can be formulated as the placement of logic gates, represented by vertices in a logic network, and the routing between gates via wires, represented as edges in a logic network, in a way that the logic of the logic network is retained in the designed circuit. Because each gate and its position in the layout is strongly dependent on the clocking inside the QCA domain, the circuit also has to follow two timing constraints to function correctly. In this section, the notation and constraints of placement and routing in the QCA domain are introduced. The P&R problem, originally derived from [67], evolves from a grid-based tile-design in conjunction with a logic network.

Definition 2.3.1. A *layout* is defined by a $w \times h$ grid $\Gamma_{w,h}$ and a graph $G(V, E)$, which is placed on the grid. Each *tile* of the layout can be accessed via its x and y coordinates. The set of tiles is denoted as T with $t = (x, y) \in T$. For any vertex of the graph, $v(x, y)$ is restricted to the boundaries $x < w$ and $y < h$. For edges $\{(x, y), (x^*, y^*)\}$ it holds $|x - x^*| + |y - y^*| = 1, 0 \leq x, x^* \leq w, 0 \leq y, y^* \leq h$.

Definition 2.3.2. A *gate-level* layout describes a layout grid in combination with a logic network $N = (\Lambda, I, \Sigma, O)$. In addition to the already known mapping *placement* p, which assigns nodes to tiles, there are two additional mappings. The *routing* r, which assigns logic network signals to layout paths (connected tiles) and a *clocking* c assigning clock numbers to tiles. The gate-level layout is therefore described as $L = (\Gamma, N, p, r, c)$.

Further, nodes placed on the gate-layout are referred to as *gates*. Two tiles $t_i = (x_i, y_i)$ and $t_j = (x_j, y_j)$ where $|x_i - x_j| + |y_i - y_j| = 1$ are called *adjacent*. A path which is wired through adjacent tiles is called *wire*. In this context, one tile corresponds to a *wire segment*. If neither a gate nor a wire segment is placed on a tile, it is empty. It follows that a layout with only empty tiles is also empty. A layout is said to be S-clocked if it follows a clocking scheme S. Otherwise, it is irregularly clocked. Moreover, an adjacent tile of a tile $t \in T$, where T is the set of all tiles, is incoming t^- if $c(t) - c(t^-) \mod clk = 1$. This means that the incoming tile can forward information to the viewed tile according to pipelined clocking. For outgoing tiles t^+ it holds that $c(t^+) - c(t) \mod clk = 1$ accordingly. For QCA it was already stated that the clock number $clk = 4$.

From this definition, there arise some difficulties of placing and routing a logic network onto a two-dimensional grid, with exception of wire crossings, which however are really costly and therefore should be minimized. One major challenge for P&R algorithms is the signal synchronization, which results in a strong dependency of clocking and signal distribution. As already pointed out, for every signal path it has to hold that information can only propagate from a tile with clocking number i to an outgoing tile with clocking number $(i + 1 \mod clk)$. This property is called the *local synchronization constraint*. The existence of possible signal paths can be assured by using predefined clocking schemes, but, however, can comprise some constraints. In addition, the *global synchronization constraint* states that every two signal paths leading to the same tile must to pass the same amount of tiles starting at their primary input. Since this constraint has to hold for every gate, the complexity increases rapidly with growing network sizes. Therefore, the combination of all these challenges forms a P&R problem, which is proven to be as $\mathcal{NP}$-complete [70].

When a gate-level layout is designed, a technology still has to be mapped onto it. To do this, the standard library proposed in Section 2.2.2 is used for the mapping. Because the definition of the P&R problem is based on the book [67], which defines it for the domain of field-coupled nanotechnologies, several FCN technologies can be mapped onto such a gate-level layout. This means that, for example, a change of clock to $clk = 3$ would also allow for placement and routing for *Nanomagnet Logic* (NML) [65]. Therefore, although the designs in this work are only for QCA, the ideas may also apply to other FCN technologies.

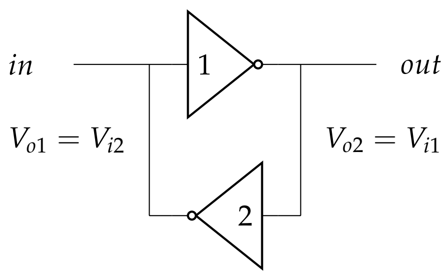


Figure 2.14: Eccles-Jordan-Flip-Flop

2.4 Sequentiality

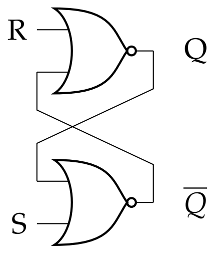
In Section 2.3 about the placement and routing problem, the main constraints for the design of QCA circuits are summarized. With this knowledge, both combinational and sequential circuits can be designed and verified. Combinational circuits can be understood as a combination of logic operations, while sequential circuits additionally include the reuse of information computed by their logic, requiring a back-loop functionality. This additional back-looping is achieved through the use of storage elements, which are built according to the clocking properties of the circuit and technology. Since clocking in QCA is completely different from CMOS, storage elements and sequentiality have to be rethought. Hence, the properties of storage elements are discussed first in the CMOS domain and then transferred to the QCA domain. It should also be noted that clocking in QCA only supports synchronous information flow, which means that in the first part, only synchronous CMOS logic is considered.

2.4.1 CMOS storage elements

In order to achieve sequential behavior in CMOS technology, *registers* are implemented into the circuits. A register is capable of storing and stabilizing several bits of information, which is then looped back to the logic via wires. In order to understand registers, first *flip-flops*, storing each one bit and their building blocks *latches* have to be discussed.

The simplest storage element in CMOS, which can store one bit, is the so-called Eccles-Jordan (EJ) flip-flop (FF). It is formed by connecting the output of one inverter to the input of another inverter and vice versa. Therefore, the logic level is propagated from one inverter stage to the other, while the same value is held in it. Due to its simplicity, a high voltage shift is needed in order to change its stored value, making the FF really inert.

Also the direct connection of the input and output makes their voltage levels highly dependent on each other, which can be interpreted as noisy behavior, also referred to as



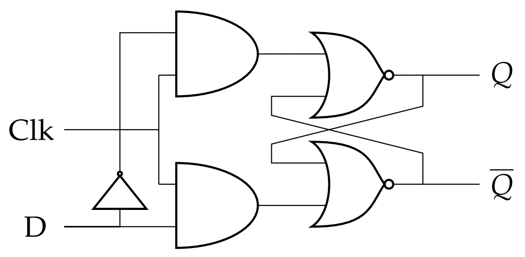
S	**R**	Q_{next}	**State**
0	0	Q	hold state
0	1	0	reset
1	0	1	set
1	1	X	undefined state

Figure 2.15: SR-Latch

transparency [23]. To avoid errors caused by the transparent behavior in the transition region, the input voltage needs to be really stable. In order to receive more robust storage elements, more sophisticated ideas were implemented while preserving the general idea introduced by the EJ-FF.

By replacing the inverters in an EJ-FF with NOR gates as shown in Figure 2.15, the transparency gets reduced and two inputs, a set S and a reset R, are introduced leading to four possible input combinations and clear states. When both $S = 0$ and $R = 0$, the current value is latched and the storage element is in the hold state. By holding $R = 0$ and changing $S = 1$, the latch output is forced to $Q = 1$, called the set state. Reversing both inputs to $R = 1$ and $S = 0$ leads to the reset state where the latch output is $Q = 0$. Furthermore, with $S, R = 1$, the latch value is unstable, and therefore we get an undefined state, prohibiting this input combination. Based on its behavior, this element is called *Set-Reset-Latch* (SR-Latch).

In order to achieve synchronous behavior the Set and Reset inputs are clocked resulting in a gated SR-Latch. To eliminate the undefined state, the set and inverted reset inputs are connected together, forming the D-Latch shown in Figure 2.16. Consequently, a value is held if the clock $clk = 0$ and the D lock propagates the input value D if $clk = 1$. To give a memory element the ability to clock Boolean operations from one stage to another, *edge-triggered* flip flops are introduced. Rising clock edges determine when the FF overwrites and passes its data. An edge-triggered *Data FF* (D-FF) can be constructed by connecting two D-latch stages behind each other. The first stage is activated at rising edges, while the second stage is activated at falling clock edges. In this way, the D-FF takes new data at a rising edge and passes them in the next clock cycle from its output. Also, the edge-triggered FF eliminates the transparency. Regarding the term edge-sensitive for D-FFs, latches are considered to be level-sensitive [23].



clk	D	Q_{next}	State
0	0	*	hold state
0	1	0	propagate
1	0	1	propagate

Figure 2.16: D-Latch

2.4.2 QCA storage elements

The goal of a storage element in QCA is to have all the properties of a D-FF, so it can store data from one clock cycle and pass it to the combinational logic in the next clock cycle. Also the element should not be transparent and the effect of an edge triggered element has to be discussed. In the QCA ONE library an effort was made to translate the D-FF into QCA by just replacing the CMOS gates and wires with the corresponding QCA gates. Thus a second external *clk* signal is introduced, mimicking the clock of a CMOS circuit. But since the QCA circuit already has its own clock with a dependency on all gates of the circuit and the clocking has some major differences CMOS, as already observed in Section 2.2.3, this implementation is questionable and the implementation of latches and FFs in QCA has to be rethought from scratch.

This leads us to ideas to look at QCA storage elements dependent on their corresponding clocking. The effort of rethinking storage elements in QCA from scratch was already made in [62]. The proposed solution to create a QCA element which is able to store one bit is rather simple. Since every tile is clocked on its own, a simple latch can be formed by a wire segment, held in the hold phase of the clocking, as shown in Figure 2.18(a). In Figure 2.17 the clocking of this wire segment is depicted. The data is propagating into the latch and by holding the hold phase by exactly one clock cycle the data is passed to the next logic block exactly one clock cycle later. Also the question arises if this element is a level-sensitive latch or if it is a edge-sensitive FF. For the latch we could argue that the information is clearly not sampled at one time point like in an ideal FF, but rather the information is taken into the latch during the whole switch phase. On the other hand one could argue that the switch phase could be seen like a real-time rising clock edge, where the data is sampled and then held while the clock is in hold phase or "1". For this work, the comparison to a FF seems to be more suited because also no transparency is allowed due to the strictly independent clocked tiles in QCA. The only functionality that the wire-FF misses is a set-reset option. The concept of replacing wire with a majority gate while maintaining the same clocking was also proposed in [62]. Now the D-FF basically has the same functionality but has

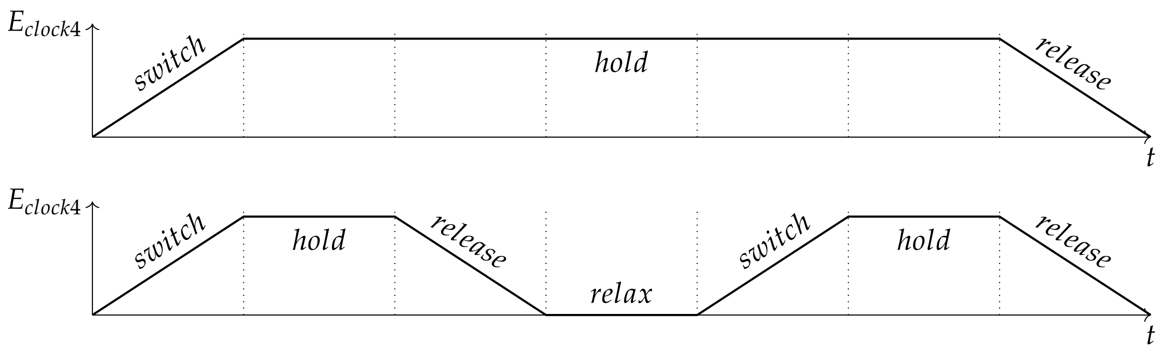
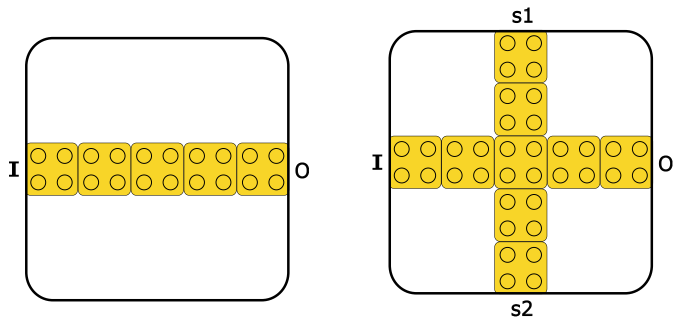


Figure 2.17: Clocking of a basic latch in QCA



(a) Wire FF (b) D-FF

Figure 2.18: QCA storage elements

two external inputs which can force the gate to zero by setting both inputs to zero or force the gate to one by setting both inputs to one. In the other configurations, the majority-FF would act as normal storage element. Figure 2.18(b) shows such a D-FF.

3 State of the Art

In this chapter, various approaches trying to solve the placement and routing problem for QCA are reviewed. They are compared to the theory provided in Chapter 2 and serve as a basis for the newly introduced P&R methodology in Chapter 4. In the first part, algorithms which are able to work only with combinational circuits are investigated under the theoretical groundwork done in Chapter 2. These algorithms are further divided into determining *optimal* and *scalable* solutions. In the second part, ideas and challenges of sequential placement and routing algorithms are investigated.

3.1 Combinational P&R Algorithms

In this section, a review of optimal and scalable solutions for the placement and routing problem is presented. However, it should be noted that optimal solutions are defined in relation to the specific problem formulation and are highly dependent on it. For large circuits, finding optimal solutions can be hindered by the high computational demands of these approaches. As such, scalable solutions, which prioritize computational efficiency over optimality, are also introduced in the latter part of this section.

3.1.1 Optimal Solutions

In order to understand optimal solutions for placement and routing, it has to be reviewed from Section 2.3 that this problem is $\mathcal{NP}$-hard. Here the complexity class $\mathcal{NP}$ (nondeterministic polynomial time) describes a set of decision problems, where problem instances with a formula, that can be evaluated to true, have a proof, that can be verified in polynomial time by a deterministic Turing machine. The existence of these problems lead to several ideas on solving them, one of these being *Satisfiability Modulo Theories* (SMT). The *satisfiability problem* can be formulated as the question if there exists a model evaluating the first-order formula over some theories to true. The consequent solving instance for propositional logic is a *Boolean Satisfiability Solver* (SAT), with its proposition being Boolean equations, that have to be proven true. With this basic instance, two different solving strategies were proposed. The first strategy is called *Eager SMT-solving* and is used for uninterpreted functions or bit-vectors, which can be derived to propositional logic. Therefore, the first step implies the transformation

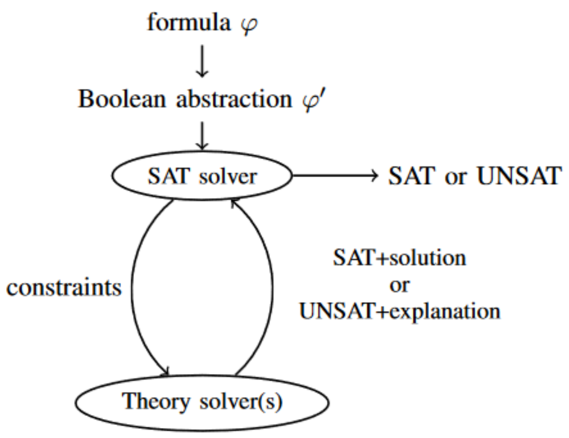


Figure 3.1: Lazy SMT-solving process [1]

of theory constraints into *equisatisfiable* propositional logic. These problem insances are then passed to SAT solvers, checking for satisfiability. Due to the equisatisfiability of the problems, a solution for the original problem can be derived from the solution of the propositional logic. The second approach called *lazy SMT-solving* refers to the assisting use of *theory solvers* and its process is depicted in Figure 3.1. In the first step the first-order problem with formula φ is transformed to a *Boolean abstraction* φ' mapping the concrete problem to an abstract problem under a set of finite Boolean predicates [4]. The abstraction is then passed to the SAT-solver, which again computes solutions and gives them to a set of theory solvers. They in turn check if the Boolean predicates hold or rather if they are consistent in the provided solution. If so, the abstraction is satisfiable and the theory solver instance returns SAT. Otherwise UNSAT with an explanation is passed back to the SAT-solver aiding the improvement of the abstraction. If the abstraction is finally found to be unsatisfied the problem is said to be unsatisfiable [1].

With this knowledge, optimal placement and routing algorithms can be discussed by reviewing two approaches proposed in [69] and [68]. The first algorithm *Exact Placement and Routing* [69] finds a valid placement, routing and clocking, also described as tuple (p, r, c), given an empty layout L and a logic network N. In order to find an optimal solution, the minimum layout size $w \times h$ has to be determined for which the constraints of (p, c, r) hold. Therefore, all possible sizes of layouts are encoded and passed to a SMT-solver iteratively and the first layout for which the solver returns true is the minimum or rather the optimal solution. The experimental results show that the determined layouts of the algorithm are many times smaller than the compared state of the art [17, 64]. But due to the complexity of the algorithm utilizing satisfiability solvers, the algorithm times out for quite small input sizes already, making it insufficient for the manufacturing of commercial QCA circuits.

The other optimal P&R algorithm proposed in [68] creates a *one-pass synthesis*, which combines logic synthesis and physical design in a single run with the idea to adapt the whole design-process to the needs of the QCA design rules. Therefore, this algorithm has to tackle two $\mathcal{NP}$-hard problems relying again on the power of satisfiability solvers. This particular algorithm uses eager SMT-solving. The idea is to eliminate some shortcomings of the two-step synthesis derived from CMOS. This includes treating wires as gates since the costs are equal in QCA and including data synchronization, which is dependent on the tiles passed. In this manner a SAT problem can be formed and passed to a SAT-solver. The instances are now created only passing an empty layout L of size $w \times h$. Even though this algorithm is able to find *truly minimal* solution since the non-optimal logic networks are eliminated, the experimental results show the same problems as in the exact P&R approach. This means that the high complexity of the satisfiability solver leads to a time-out of the algorithms for circuits with a gate size $|N| \geq 30$.

3.1.2 Scalable Solutions

These shortcomings lead to the usage of scalable placement and routing algorithms. This approach trades optimality of the circuit for computing time, yielding larger, more expensive layouts, but in short time. This makes them scalable in the time domain and therefore applicable for the manufacturing of commercial QCA circuits. All algorithms reviewed in the following are based on the original VLSI process, meaning that they treat logic synthesis and physical design as their own problem and not as one-pass synthesis.

Preprocessing

Starting with logic synthesis, many works on QCA present a preprocessing of logic networks enabling them to be translated directly into gate-level representations. There are several steps which are widely used to modify logic networks. The first of them is the node duplication or rather dummy node insertion. The idea of this process is to minimize wire-crossings, which we have analyzed to be very costly in QCA, and to reduce the number of fan-outs at the nodes, leading to a reduction of the place and route complexity. One simple algorithm for this is to visit every node in a breadth-first search from each primary output to the primary inputs. If the current node hasn't been visited, it is marked as visited and if an already marked node is visited it is duplicated. This process is quite problematic, because not only the visited node is duplicated, but

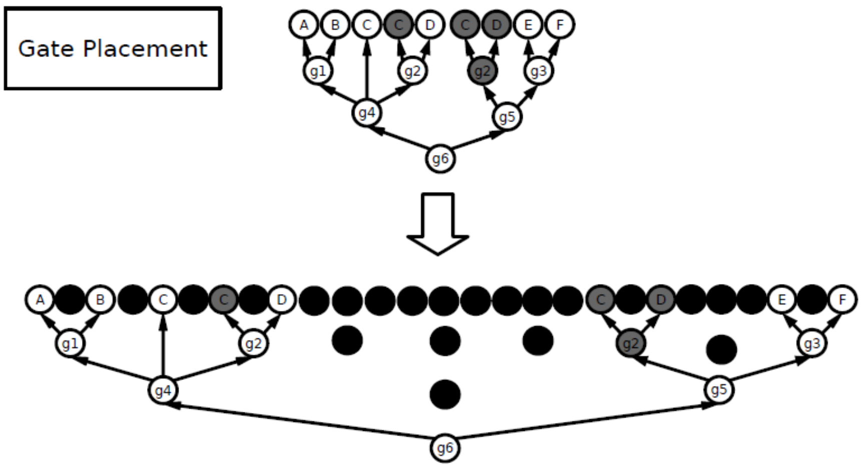


Figure 3.2: Gate placement with black circles showing wasted area [59]

also all the nodes included in the sub tree rooted by it [59]. From this simple example it can already be suggested that the insertion of dummy nodes can lead to uncontrollable growth of the logic network and also layout size. More dedicated algorithms do not have such a high overhead in dummy nodes but in contrast they cannot eliminate all wire crossings, making it necessary to include nodes for crossings, so-called *crossing edge insertions* [15]. Another preprocessing step, including the insertion of so-called *buffer nodes*, aims to synchronize signals in order to meet the global timing constraint. Since this constraint requires two paths leading to the same node to pass the same amount of tiles, a valid layout can be easily deduced from the logic network, if every path has the same amount of nodes [12]. Some approaches insert even a higher number of nodes in order to obtain a complete ternary tree. This idea is based on the majority function representation of gates. When extra nodes are included in the logic network, also extra area is produced as shown in Figure 3.2 and this in turn leads to an increase in wire lengths. Because these approaches are based on cell-based clocking, this implies that if the longest wire has to be split into more than one clock zone also the shortest wire has to be split into the same amount of clock zones in order to preserve the signal synchronization for gates with two or three inputs, which again produces area overhead [59].

Another big problem of these algorithms is the requirement of cell-based clocking itself, which has already been shown to be unrealistic. Even though there exist algorithms using tile-based clocking they are limited by the general drawbacks of preprocessing [64], leading to exploding logic networks and even the use of greedy placement and routing algorithms limiting the approach to small and simple reconver-

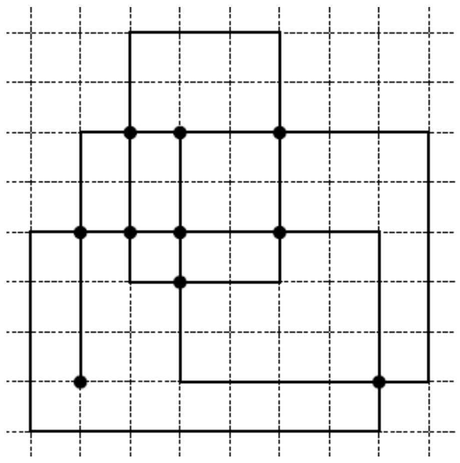


Figure 3.3: Example OGD drawing [16]

gent patterns [59].

Ortho Algorithm

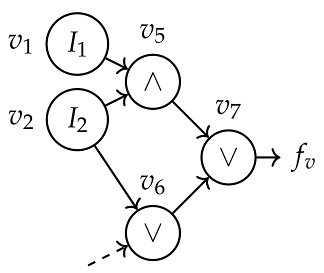
All these reasons lead to the proposal of *ortho*, an algorithm implementing a scalable placement, routing and clocking without preprocessing steps proposed in [71]. Since this algorithms forms the base of this work, the algorithm is explained in detail in the following.

First of all, a proper representation of the P&R problem is needed. Therefore, in some works already the idea of an orthogonal embedding, had been proposed [12]. Orthogonal embedding is the mapping of a logic network onto a two-dimensional grid, so it can be seen as an assignment of the tuple (p, r). For ortho, this is done by orthogonal graph drawing (OGD), which is described in [16].

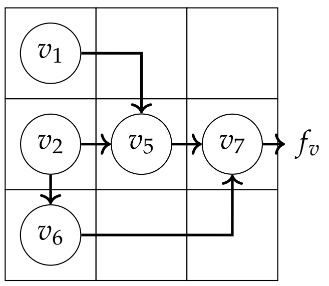
Definition 3.1.1 (Orthogonal Graph Drawing)**.** OGD maps a graph $G = (V, E)$ onto a plane grid with size $w \times h$. The mapping assigns vertices $v \in V$ with coordinates (x, y) to grid points, with $1 \leq x < w$ and $1 \leq y < h$. Edges $e \in E$ are assigned to paths in the grid, so they consist only of horizontal and vertical segments. The paths are non-overlapping, meaning that they are not allowed to cross any vertices.

Figure 3.3 shows an example of OGD. The dots in the graph represent vertices and are connected via straight line paths. Thus, the graph is drawn orthogonally.

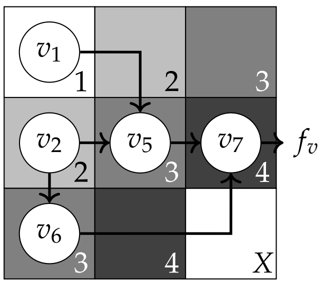
Nonetheless, OGD only respects the placement and routing, leaving the clocking to be addressed. The problem of insufficient clocking in a valid OGD representation can be seen from the example in Figure 3.4. For the given logic network in Figure 3.4(a) and its



(a) Partial representation of a logic network



(b) OGD representation of the logic network



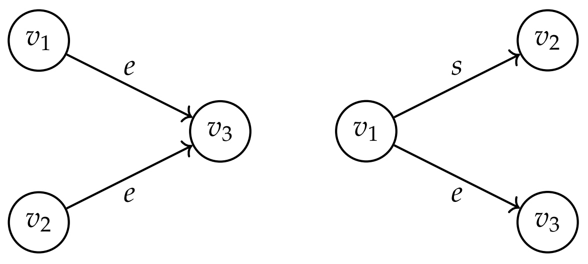
(c) No sufficient clocking for this OGD can be examined

Figure 3.4: Insufficient timing constraints of an OGD representation [67]

valid OGD in Figure 3.4(b), there is no clocking which can resolve the synchronization constraints. In Figure 3.4(c) it stands out that for the down right corner no clocking zone can be found so that either the local synchronization constraint but also the global synchronization constraint are satisfied. Since the clocking or rather signal synchronization was a main task of the preprocessing, which is not used here, some other solution has to be found.

The idea used for the ortho algorithm comes from an extension to OGDs, which allows to determine a special OGD from a logic network in polynomial time being; the constraint needed for a scalable approach. The base used in [71] is formed by Therese Biedl [8], who proposed an OGD with an additional edge coloring. Although the effectiveness and complexity bounds in her work were proven on the restriction of *undirected 3-graphs*, and as we already examined from the previous chapter, a logic network is neither containing only nodes of most degree 3 nor undirected. To overcome the fist restriction, a custom logic network can be created by assigning dedicated nodes for fan-outs and inverters. Also complex gates like majority gates get decomposed. This way, the maximum node degree gets decreased to three, while the expressiveness of the logic network representation is maintained. The second restriction can be overcome by a custom coloring built on the original approach, which also serves as direction assignment. Given a logic network converted to a 3-graph, the coloring in form of edge directions $d : \Delta \to \{east, south\}$ is assigned. The coloring can be understood as relative position arrangement. If an edge (v_i, v_j) is colored *east*, it means that the vertex v_j is positioned east of v_i, so that $x_j > x_i$. The color *south* for an edge (v_i, v_j) assigns v_j a relative position of v_i, so that v_j is south of v_i or $y_j > y_i$. In order to color a graph with only these two colors the following *assignment constraints* must hold:

1. All **incoming** edges of a vertex have to be painted with the **same** color.



(a) Color assignment of incoming signals

(b) Color assignment of outgoing signals

Figure 3.5: Relative positions of an OGD graph with correct color assinment

2. All **outgoing** edges of a vertex have to be painted with **different** colors.

The relative position assignment under the proposed constraints can be seen at an arbitrary example in Figure 3.5. In the example for outgoing edges (Figure 3.5(a)), the assignment constraint makes sure that two outgoing edges of the same vertex are routed in different directions to avoid a conflict. Equivalent to the definition of the colors, the layout is increased in x-direction for an east-coloring and analogously extended in the y-Direction for a south-coloring. Figure 3.5(b) depicts the assignment constraint for the incoming edges of one vertex. This assignment has to use one color and the node can be set non-conflicting for both incoming nodes in the layout by extending it in x-direction based on the east-coloring. However, there exist logic networks for which no coloring in regards to the constraints can be found. When such a coloring conflict appears, the conflicting edge, for which no direction can be assigned after the formulated constraints is divided and an auxiliary node is introduced resolving the conflict. In order to allow data to propagate, ortho needs to map a valid clocking onto the layout. Because ortho uses OGD with exactly two directions *east* and *south*, the 2DDWave scheme, which also supports the data flow in exactly two directions, is perfectly suited for the algorithm. Also the usage of a predefined clocking scheme already gives a solution for the local synchronization constraint and due to the uniformity and simplicity of the 2DDWave scheme also the global synchronization constraint for nodes placed and routed after the proposed direction assignment is maintained.

In the following the pseudo code of ortho, derived from [71] is depicted as Algorithm 1 and described in own words, before it is evaluated on an example and its main characteristics are described. Following the VLSI design process the ortho algorithm has as input a logic network N and a clocking scheme or rather a clock number *clk* in order to fulfill the timing constraints. As already mentioned, N has to be converted into a 3-graph by substitution such that a valid coloring can be assigned. The corresponding

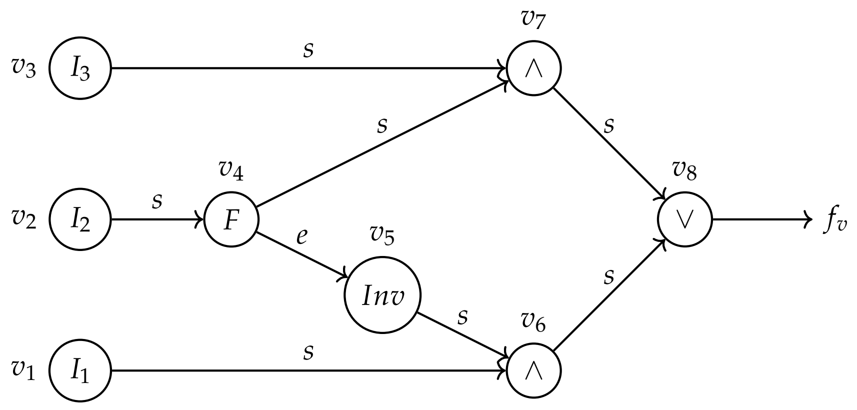


Figure 3.6: Logic Network of a 2:1 MUX after fan-out substitution, inverter insertion and coloring

logic network for a 2:1 MUX can be seen in Figure 3.6. An empty layout L with a 2DD-Wave clocking scheme is created and the coloring is calculated for N. The nodes need to be topologically ordered starting with the lowest number at the inputs and the highest numbers at the outputs. Therefore, the algorithm is starting with the lowest numbered vertices representing primary inputs. In order to connect the inputs to external signals, they are placed at the borders of the layout. In ortho, the first column of the layout is therefore reserved for placing the inputs. Due to the properties of OGD, a conflict arises, when inputs have outgoing edges colored south, because the algorithm would then wire these edges into y-direction and therefore over other primary inputs. Since this is forbidden in OGD and for QCA layouts, these conflicts need to be resolved by the algorithm. To do so, primary inputs colored *south* are resolved by first rewiring them each on a new column, allowing a wiring into y-direction and thus the placement of the nodes connected to their outgoing edges. Further, for the placement and routing of all nodes in the logic network, the two parameters coloring and the updated parameter (w, h), saving the current dimensions of L, need to be evaluated in each step. If a node is colored *east*, the layout is extended by one column and the node is placed at $(w-1, h_p)$, where w is the current width of L and h_p is the maximum vertical position of its parents. According to this scheme for nodes colored *south*, the layout is extended by one row, and the node is placed to $(w_p, h-1,)$, where w_p is the maximum horizontal position of the node's parents and h is the current height of the layout. After placing the node, it is wired to its predecessors, while the placement into a new row or column makes sure the wiring does not pass over another gate. If only one predecessor exists the wiring also goes only *south* or *east*. If two predecessors exist, in case of *east* the predecessor defining h_p is also only wired in x-direction, while the

Algorithm 1 Ortho algorithm

```
    Input: Logic network $N$
    Input: Clock number $clk$
    Output: Gate level layout $L$
 1: Convert $N$ to a 3-graph by substitution
 2: $L \leftarrow$ empty 2DDWave-clocked layout of size $(w = 0) \times (h = 0)$
 3: Generate direction assignment $d : \Sigma \rightarrow \{east, south\}$ and subdivide signals if
    necessary
 4: Compute topological ordering $v_1, ..., v_i \in N$
 5: Extend $L$ by one column and reserve it for primary inputs
 6: for each vertex $v_1, ..., v_i \in N$ with at most two incoming signals $\sigma_1, \sigma_2$ do
 7:     if vertex $v$ is terminal/primary input then
 8:         Extend $L$ by one row
 9:         Place $v$ at position $(0, h - 1)$
10:         if vertex $v$ is colored $south$ then
11:             Extend $L$ by one column
12:             Wire the primary input to position $(w - 1, h - 1)$
13:         end if
14:     else if $d(\sigma_1) = d(\sigma_2) = east$ then
15:         Extend $L$ by one column
16:         $h_p \leftarrow$ max. vertical position of $v$'s predecessors
17:         Place $v$ at position $(w - 1, h_p)$
18:     else if signals are labeled $south$ then
19:         Extend $L$ by one row
20:         $w_p \leftarrow$ max. horizontal position of $v$'s predecessors
21:         Place $v$ at position $(w_p, h - 1)$
22:     end if
23:     Draw orthogonal wire segments to connect $v$ with its predecessor(s) accordingly
24: end for
25: Connect the primary outputs to the respective borders
26: return $L$
```

other predecessor has to be wired with two wire segments, the first also going into x-direction and the second one connecting in y-direction. If the node is colored *south* the wiring goes south first and then east. After all nodes were placed in this fashion the primary outputs are also connected to the borders either to the east or the south and the finished layout L is returned by the algorithm.

The example depicted in Figure 3.7 shows the (p, r, c) of a 2:1 MUX. As a reminder, the logic network is depicted in Figure 3.6. Starting with a layout of size $(1, 0)$, for the PI v_1, the layout is extended by one column to $(1, 1)$ and is placed to $(0, h - 1) = (0, 0)$. Because the outgoing edge of the PI is labeled *south*, the wiring has to be resolved by extending L to $(2, 1)$ and wiring the PI to $(w - 1, h - 1) = (1, 0)$. For the other PIs v_2, v_3 it follows the placement to $(0, 1)$ and $(0, 2)$ and an increase of one column per PI since they have both outgoing edges labeled south. The resolving wiring leads to $(2, 1)$ and $(3, 2)$. Now the remaining nodes can be placed following the $east, south$ scheme. The size of the layout after placing the inputs and wiring them to non-conflicting tiles is (4×3). With v_4 a fan-out node is placed south of the third input, which has coordinates $(3, 2)$. After extending L by one row, the y-coordinate is evaluated to be $w - 1 = 3$ and the x-coordinate 3 is adopted from its only predecessor. Therefore, the node is placed on $(3, 3)$ and L has size $(4, 4)$. In the same fashion the parent node of the fan-out v_5, representing an inverter is placed east of it. The coordinates of the inverter are $(4, 3)$, while L has size $(5, 4)$. Looking at the next node v_6, which is the first node with two parents and which is labeled south now the x-coordinate is determined by the eastern predecessor, so v_5 at $(4, 3)$. The y-coordinate is again adapted form the size of the layout, after it was increased in y-direction once, resulting in a placement on $(4, 4)$. The same way the AND-node v_7 at $(3, 5)$ and the OR-node v_8 at $(4, 6)$ are placed. When all nodes are placed, the primary output has to be placed from v_8. For this the layout is increased by one column again to the size (6×7) and the PO is placed at the eastern border.

From the returned layout we can see that the signal flow is straight forward in south-eastern direction due to the 2DD-Wave clocking scheme the algorithm is bound to and as already discussed in Section 2.2.3 about clocking, this limits ortho in many ways. First of all, due to the selection of the "+"-majority gate as standard gate, they cannot be placed on the 2DD-Wave scheme, because the clocking only allows two-input logic gates. Also back-loops are not allowed in the clocking, prohibiting the placement and routing of sequential circuits. However, ortho constitutes an efficient tool for the placement and routing of purely combinational circuits. As shown in paper [69], the 2DD-Wave scheme is the most area efficient clocking scheme when it comes to combinational circuits, beating both the USE and RES scheme.

Ropper and Migortho

With a deep understanding of the ortho algorithm with its constraints, involving drawbacks and advantages, other ideas with comparable approaches or based on the ortho algorithm can be discussed. *Ropper*, a placement and routing framework [18] proposes an algorithm, which is based on [64]. As already discussed in the part about preprocessing, this algorithm brings some disadvantages, because dummy nodes are

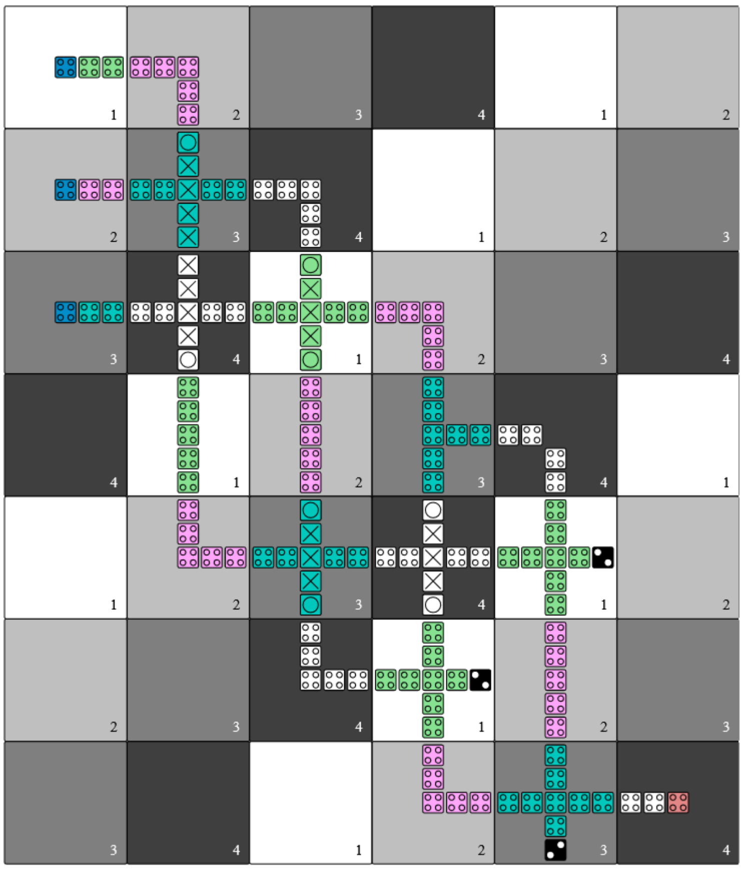


Figure 3.7: Placement and routing of a 2:1 mux network using the ortho algorithm

inserted as part of logic synthesis, leading to an increased size of the logic network. Nevertheless, the authors of Ropper point out that they have overcome some restrictions of ortho, one of them being able to place majority gates and also a more area efficient placement and routing. But these improvements come at a price. First of all, the framework only achieves the placement and routing of majority gates not because of a clocking scheme providing three input tiles to a given tile, but supporting the use of rotated majority gates, which had been discussed to be very prone to crosstalk. Also custom gates and double wiring is used, so that many tiles have QCA cells placed only with a distance of one cell; unlike [72] suggests a minimum distance of two QCA cells. The use of these custom gates is necessary in this algorithm, because the used design does not rely on the same strict constraints as ortho. The Ropper framework even routes wires above gates (solved with custom tiles) and does not use border inputs and outputs making it challenging to input and read data from the circuit. Based on the argumentation in this work, the Ropper framework violates the design rules, which have been analyzed to be necessary for a sufficient placement and routing algorithm [18].

Another paper trying to implement majority gates for QCA is *migortho* [27], which is based on the ortho algorithm. The difference of the algorithms is the use of the underlying gate-library. Migortho utilizes the QCA ONE library. From the preliminaries, it is already known that rotated majority gates are used and, therefore, also double wires need to be used. This means that circuits designed by migortho are also being considered to be prone to crosstalk, following the argumentation from above. But the algorithm shows that with the use of a different library, ortho is already powerful enough to overcome some restrictions.

In light of the limitations and shortcomings of ortho and other state-of-the-art algorithms, this work aims to investigate and implement various techniques based on ortho to improve its functionalities. To achieve this, an input ordering is introduced to the logic network to minimize wire crossings and reduce the area required. Additionally, the implementation of "+" majority gates is proposed as a means to decrease the total number of gates and wire crossings. Currently, only rotated majority gates can be implemented in automated placement and routing. Ultimately, an automated strategy for the placement and routing of sequential circuits is developed using the ortho approach.

3.2 Design of Sequential QCA circuits

In this section, the state of the art for sequential circuit design in QCA is discussed. Compared to the algorithms existent for the placement and routing for combinational logic, this area is still in its infancy. This section first focuses on the main part of implementing sequential logic and then uses this knowledge to give an insight into the implementation of storage cells.

3.2.1 Sequential logic in QCA

In recent literature, several attempts have been made to implement latches and flip-flops (FFs) in order to obtain storage elements and enable sequentiality for quantum-dot cellular automata (QCA) circuits [36, 47, 56, 60]. These works primarily aim to translate Boolean CMOS equations into majority representations and implement this representation using QCA gates. Despite the use of 4-phase clocking in QCA circuits, the resulting circuits often rely on external 2-phase clocking signals adopted from the CMOS domain, as reported in [36]. The authors of [47] claim that latching can be achieved using QCA clocking, but this would restrict the circuit's functionality. However, as discussed in the sections on clocking (2.2.3) and storage elements (2.4), it is suggested that the opposite is true and that the external clocking signal is unnecessary as the 4-phase clocking of QCA circuits can accomplish sequential functionalities without restriction.

While some works propose more advanced sequential circuits, such as dual edge-triggered D-FFs [56] and reversible latches [60], they still rely on cell-based clocking, which was found to be insufficient in Section 2.2.3. The USE clocking scheme [13] has been proposed as an evolution towards sequential tile-based placement and routing, but leads to worse layouts compared to 2DD-Wave for most combinational benchmark problems [69]. Other works, such as [41] and [7], proposed different latches and sequential element implementation algorithms using the USE scheme. However, these approaches all share the drawback of translating sequential logic directly from the CMOS domain.

As previously discussed in this work, wire segments are proposed as storage elements as suggested in [62]. But, it has to be mentioned that in [62] the D-FF is introduced as synchronization element (SE). The SE is used to build delays on certain positions or tiles of the layout in order to obey the synchronization constraints. In this work, storage elements should be used for implementing sequential circuits. Therefore, the basic idea of this approach can be used, but is modified. Also, storage elements are examined in the context of placement and routing, and the natural delay arising from routing is utilized to construct these storage elements. This approach eliminates the need for custom clocking, which is currently necessary to build storage elements.

3.2.2 QCA storage cells (QCA RAM)

Another area of research in QCA technology focuses on the implementation of random access memory (RAM) cells. While this work primarily focuses on a placement and routing algorithm for sequential logic, the implementation of QCA storage can also be adapted. The current state of the art can be divided into two approaches for implementing RAM in QCA.

The first approach, as presented in the papers [14, 72, 57], involves translating CMOS technology into QCA and thus dealing with the same shortcomings resulting from an external clock signal. Additionally, these circuits are also based on cell-based clocking, which is insufficient for this work.

The second approach, as presented in [2, 38], proposes the use of MUX structures to implement RAM cells. By using a 2:1 MUX with a RAM cell holding one bit of information and a bitline (BL) also holding one bit as inputs, the 2:1 MUX can decide whether the information on the BL is passed into the RAM cell or if the information in the RAM cell is retained, based on the information on the wordline (WL) which serves as the third input to the 2:1 MUX. However, the implementations shown in [2] and [38] still use an external clock signal and are also clocked cell-based.

The ideas of storage elements in QCA can be combined with the use of MUX to build a RAM cell, which can be constructed using ortho and the corresponding Sequential Distribution Network, as proposed later in this work.

4 Distribution Networks for Scalable P&R

This chapter proposes three different signal Distribution Networks (DNs) that overcome the restrictions of the placement and routing algorithm, ortho, which was reviewed as the state of the art in Chapter 3. One restriction of ortho is the big area usage and the high number of wire crossings. To reduce these shortcomings an Ordering DN (ODN) is proposed in this thesis. The second big restriction ortho has, is that majority functions need to be decomposed into multiple gates, even though QCA technology provides a majority gate. In this thesis a Majority Gates DN (MGDN) is introduced, which enables ortho to make use of this theoretical advantage. Also ortho is not able to place any sequential circuits. Therefore, a Sequential DN (SDN) is introduced.

The placement and routing algorithm, ortho, has been shown to have limitations regarding area usage, wire crossings, majority gates and sequentiality, but it still possesses a powerful placement and routing procedure. To overcome these limitations, it is proposed to maintain the foundation of the algorithm while incorporating new functionalities. The functionalities include an ordering of inputs and the placement and routing of majority gates and sequential QCA circuits. However, these new functionalities often result in irregularities, such as in the clocking or routing, and the goal of the proposed signal DNs is to redistribute signals on the layout in a way that these irregular parts align with the regular placement and routing while still satisfying all design constraints. In this way, the algorithm can still function in a similar manner as ortho. However, the underlying logic network and the placement and routing must also be modified to incorporate the desired functionalities, adding complexity to the preprocessing and algorithm.

Since ortho is restricted to the use of only 2DD-Wave clocking, the signal DNs must have the ability to change the clocking in the layout to implement their respective functionalities, regarding the implementation of majority gates and sequentiality. Therefore, the implementation of these networks must be done with care and synchronization constraints must be closely considered.

The ODN is discussed first, which aims to reduce the area in the input region of the layout. Afterwards, the networks used for implementing majority gates and sequential parts are discussed and analyzed.

4.1 Ordering Distribution Network (ODN)

In this section, we propose the use of an Orthogonal Data Network (ODN) within ortho for the scalable placement and routing of QCA circuits. The goal of the ODN is to minimize wire crossings and maximize use of space by ordering and placing inputs in a specific way. As shown in Figure 4.1, the ODN significantly reduces area usage and the number of wire crossings when compared to a traditional orthogonal design. This is achieved through two main concepts: (1) reordering of primary inputs and (2) utilizing the input area for gate placement. In Figure 4.1(a) and Figure 4.1(b) the PIs and the respective reordered PIs are marked blue and the input area is highlighted red. These concepts are further explained in detail in the following sections, and the implementation of the ODN requires four steps to be integrated into the ortho algorithm. These include the ordering of primary inputs (Section 4.1.1), the implementation of conditional coloring (Section 4.1.2), the introduction of a new rule for placement and routing (Section 4.1.3), and the application of inverter balancing (Section 4.1.4).

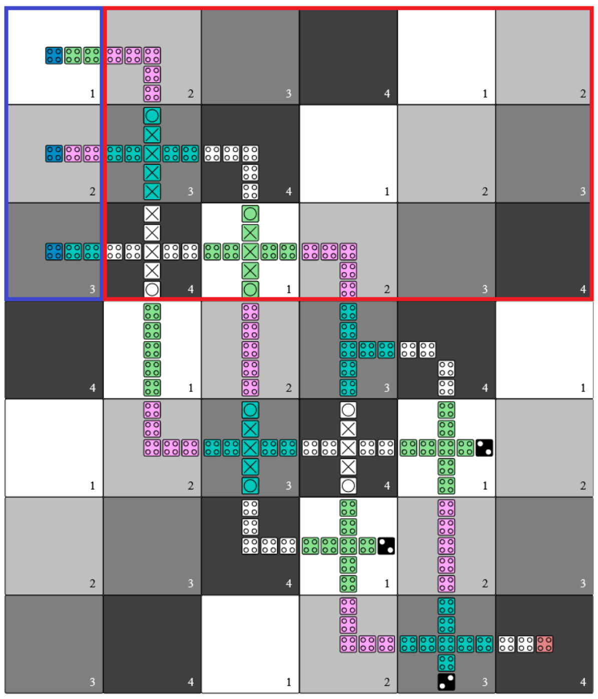
4.1.1 Ordering of PIs

In this section, we discuss the ordering of PIs in our design. Considering an AND gate connecting two inputs, it is ideal for the two PIs to be placed directly under each other in order to reduce the amount of wiring needed. When the two PIs are placed far away from each other, the distance for wiring and the probability of wire crossings increases. This highlights the importance of proper PI ordering in reducing wiring and minimizing wire crossings.

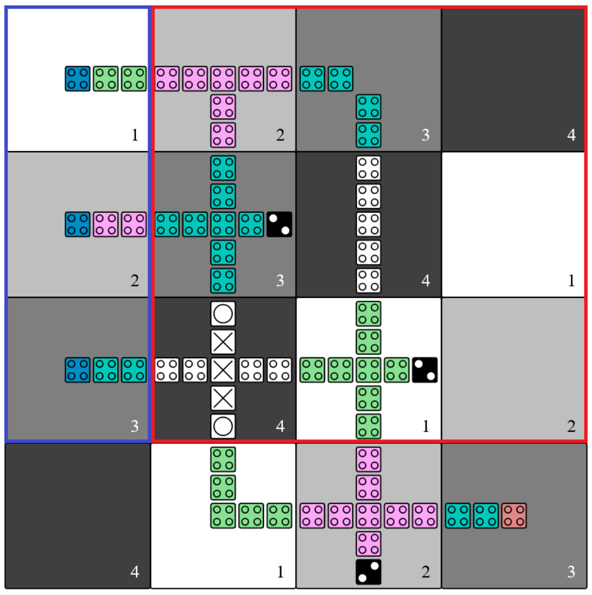
To examine the concept, it is necessary to first identify the part of the logic network that is considered for the ODN. This can be determined by examining the nodes that are placed in the input area. These include nodes that are directly connected to the PIs and nodes that are connected to these nodes and colored *east*. For the network, all nodes that are successively connected, starting at each PI and ending at the first node with two fan-ins, are considered. Figure 4.2 shows this principle. Once these nodes are placed correctly, all conflicts are resolved, and the ortho algorithm can function as intended. The schematic of a resulting layout is depicted in Figure 4.3.

For the ordering, PIs connected to the same two-fan-in gate are placed consecutively, first placing PIs connected to fan-outs and minimizing the routing distance, thus reducing the probability of wire crossings. Figure 4.2 shows that all three PIs are connected over the two AND gates. First the PI connected to the fan-out is placed. Second the PI with the branch without an inverter is placed, and then the PI with the inverter in its branch.

Additionally, after reordering the PIs, the remaining logic network is ordered topo-



(a) Placement and routing of a 2:1 MUX network using the ortho algorithm



(b) Placement and routing of a 2:1 MUX network using the ortho algorithm with the ODN

Figure 4.1: Comparison of ortho and ortho including ODN

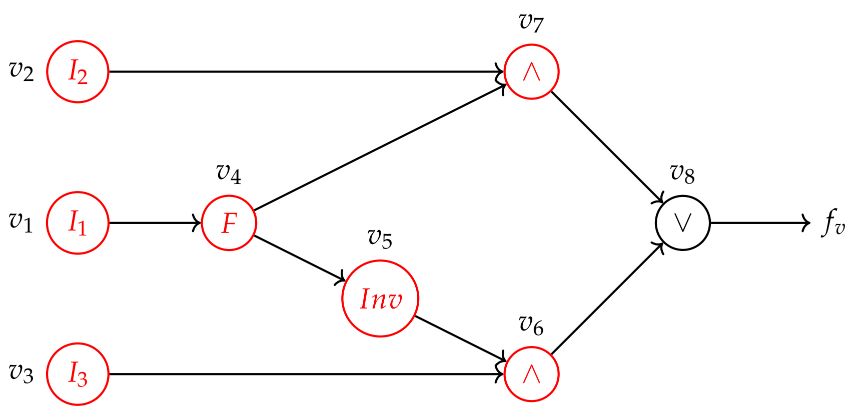


Figure 4.2: Logic Network of a 2:1 MUX with gates viewed in the ODN

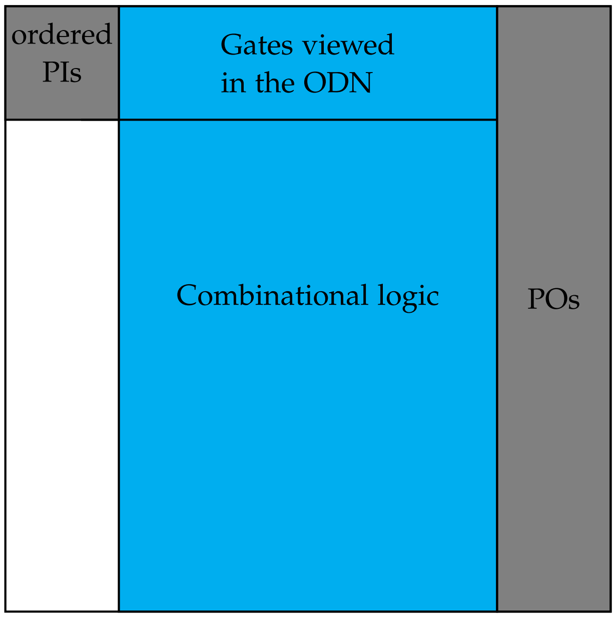


Figure 4.3: Scheme of area usage in the layout using the ODN

logically, so that the improvements following the ordering of the PIs can be applied to the entire network. In the following the steps allowing the use of the input area are proposed. Therefore, first the conditional coloring and then a new P&R rule are discussed.

4.1.2 Conditional Coloring

In this section a conditional coloring is proposed, which finds a valid coloring for the gates viewed by the ODN and allows a placement and routing of these gates without resulting in conflicts. Recalling the pseudo-code from the ortho Algorithm 1, an input has a conflict when the gate connected to it is colored *south*, which would lead the routing to wire over the other inputs in the same column, which is not allowed. This means that the area overhead in the input region is highly dependent on the coloring assigned to the outgoing edges of the inputs. Line 3 of the pseudo-code of ortho, which invokes the coloring algorithm, finds a valid but not an optimal coloring for the given logic network. Unfortunately, due to the algorithm's nature, it often assigns the color *south* to edges connected to PIs resulting in conflicts and area overhead.

In the following, the conditional coloring is discussed regarding different gate-types, which can be connected to PIs. Figure 4.4 shows the conditional coloring chosen for the logic network of a 2:1 MUX. It has to be noted, that the chosen coloring already considers the new rule for PIs with edges connected to gates colored south. The direction assignment for one-fan-in nodes, including inverters and fan-out nodes, can be chosen arbitrarily as the PI to which they are connected has only one outgoing edge, resulting in no dependencies. In this case, the non-conflicting *east* assignment can always be chosen. When examining two-input logic gates such as AND and OR gates, the coloring can only be chosen arbitrarily if both incoming nodes are PIs, again allowing for the non-conflicting assignment of east. In all other cases, the direction assignment must consider the coloring of the other incoming edge of the gate. To determine dependencies, all one-fan-in nodes must first be colored, including inverters and fan-outs.

Regarding fan-outs, following the coloring rules, the two outgoing edges need to be colored in different directions, so that the fan-out gates placed into the network have one output assigned with color *east* and one output assigned with color *south*. Considering that the second coloring constraint requires both incoming edges of the gate connected to the edge colored *south*, also to be colored *south*, and the second incoming edge being connected to a PI, we again get a conflict and we can see that a conditional coloring alone is not powerful enough to resolve all conflicts.

To enable the use of the input area, additionally a new rule for edges colored *south* is introduced, making the rewiring redundant.

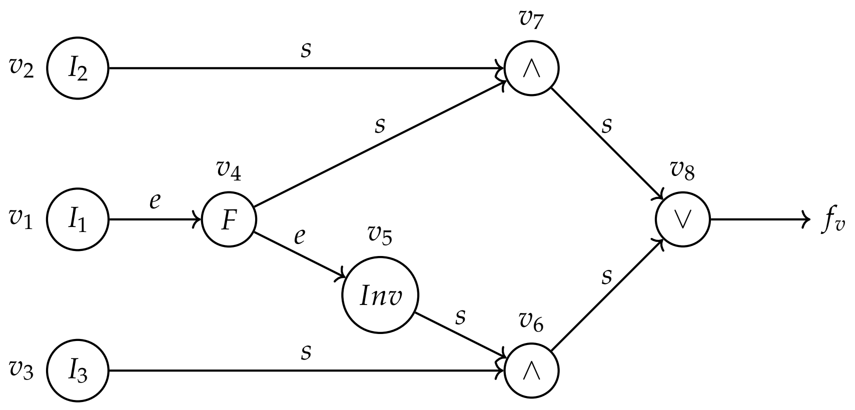
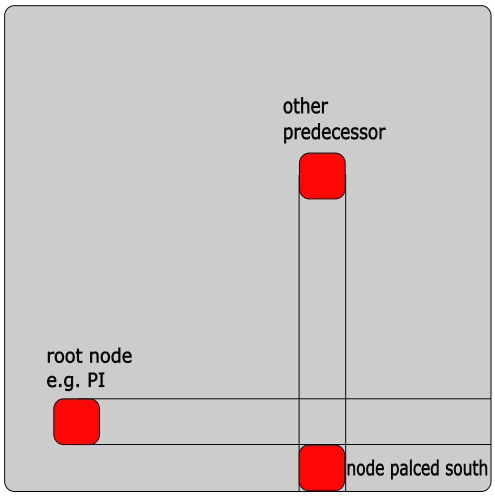


Figure 4.4: Logic Network of a 2:1 MUX with conditional coloring

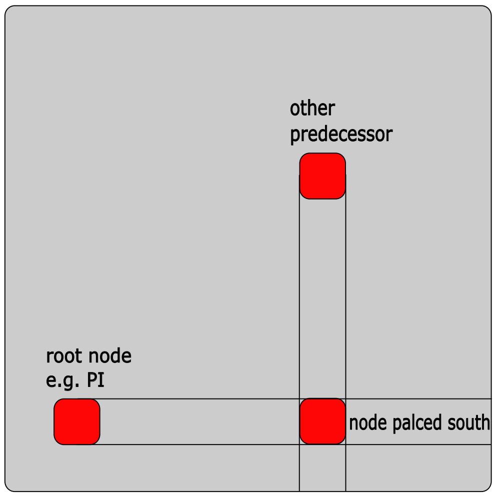
4.1.3 New P&R rule

In this section, a new P&R rule for conflicting gates in the ODN is proposed. The conflicting gates are the gates connected to PIs and colored *south*. The new rule is found to be effective not only inside the conflicting input area, but for the whole layout.

Looking at the original ortho algorithm part (lines 14-22) handling the placement of nodes based on their coloring makes sure that every gate placed *east* occupies a new column and every node colored *south* occupies a new row. These placement rules allow every gate to be placed without interfering with other gates, but the rules have been found to be too restrictive, allowing the following placement rule for *south*. Figure 4.5 shows the *south* rule of ortho and the *south* rule introduced by the ODN. If a node is labeled *south* and its predecessor, which has the lower horizontal position **also** has the higher vertical position, it is called *root node* and the layout is **not** extended by a row while the gate is still in position $(w_p, h-1)$. Following this rule the gate is now placed in the same row as its root node and the same column as the predecessor with the higher x-coordinate. If we apply this to a two-input gate in the ODN with a PI and a fan-out node as predecessors, the PI is always the root node due to the ordering and new coloring. Thus, the new rule allows the two-input gate connected to the PI colored *south* and the fan-out node to be placed in the same column as the PI, resulting in no conflict because the node is not *actually* placed *south* of its predecessors. It was found that this rule could not only be utilized for resolving PI conflicts, but also for the general *south* placement in the algorithm with one exception. Though, considering a fan-out node to be the root node, the coloring would wire both the eastern and the southern colored outgoing edges onto the same row, yielding a conflict. For this case the new rule is not applied and this case is excluded for the input area through the



(a) Placement rule of ortho



(b) Placement rule introduced with ODN

Figure 4.5: Schematic representation of rules when placing nodes colored *south*

ordering. The resulting pseudo-code snippets replacing the used code are shown in Algorithm 2.

Because this new rule can be applied not only for the input area but for all two-fan-in logic gates placed *south* in the layout, area can be saved also for gates not viewed in the ODN.

4.1.4 Inverter Balancing

In this section a inverter balancing is introduced with the goal to reduce the number of inverters in the logic network. Considering an inverter node is assigned *south*, such as after a fan-out node, and it is intended to be placed in the same row as a PI, a conflict arises because the input always has to wire in the x-direction. To minimize conflicts, all inverters colored *south* must be placed at a minimum of one row further than the most southern PI.

To further reduce the number of inverters in the logic network and prevent excessive overhead and preserve all the advantages of the presented approach, an inverter balancing is introduced. This aims to reduce the number of inverters by substituting them at fan-out nodes. If a fan-out node has two inverters connected to its outgoing edges, these inverters can be replaced by a single inverter as the incoming node to the fan-out, resulting in an overall lower number of inverter nodes and preventing inverters in the ODN to be colored *south*.

Algorithm 2 Ortho changes with the ODN

```
⋮
  Convert $N$ to a 3-graph by substitution and balance inverters at fan-out nodes
  Order PI nodes
  ⋮
  Generate **conditional** direction assignment $d : \Delta \to \{east, south\}$ and subdivide signals if necessary
  Compute topological ordering $v_1, ..., v_i \in N$
  Extend $L$ by one column and reserve it for PIs
  for each vertex $v_1, ..., v_i \in N$ with at most two incoming signals $\sigma_1, \sigma_2$ do
      if vertex $v$ is terminal/PI then
          Extend $L$ by one row
          Place $v$ at position $(0, h-1)$
      else if $d(\sigma_1) = d(\sigma_2) = east$ then
          ⋮
      else if signals are labeled $south$ then
          if not root node exists then
              Extend $L$ by one row
          end if
          $w_p \leftarrow$ max. horizontal position of $v$'s predecessors
          Place $v$ at position $(w_p, h-1)$
      end if
  end for
  ⋮
  return $L$
```

Looking again at Figure 4.1(b), the placement and routing of the ortho algorithm after implementing the proposed ODN can be seen. The ordering of the inputs first places the fan-out node and then the two connected PIs. Since the inverter is part of the ODN it gets colored *east* and allows the AND gates connected to the PIs to be colored *south* and the new rule for placement and routing can be applied. The last OR gate is not part of the ODN and is therefore placed by the default rules of the ortho algorithm. In comparison to the layout in Figure 4.1(a), it can be quickly seen that the resulting layout saves up area and even wire crossings. The detailed results are presented and analyzed in the next chapter.

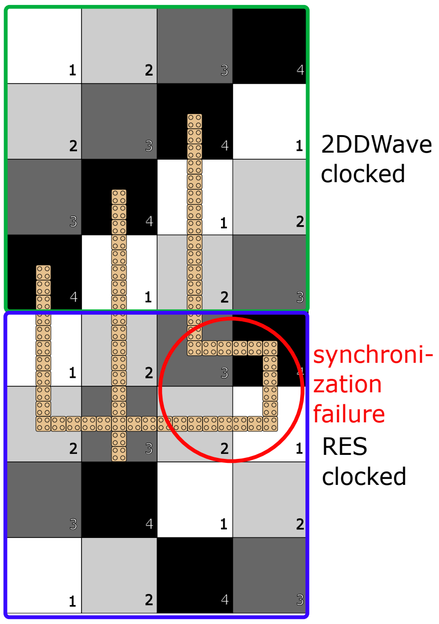


Figure 4.6: Global sychronization violation when connecting 2DD-Wave and RES clocked scheme

4.2 Majority Gate Distribution Network (MGDN)

This section addresses the placement and routing of majority gates in QCA circuits using the ortho algorithm. The use of majority gates is significant in QCA as they can implement the majority function using only one gate, unlike in CMOS circuits which require multiple gates, as defined in Definition 2.1.3. However, this theoretical advantage can only be realized through efficient placement and routing. To this end, a majority gates DN (MGDN) for the orthogonal algorithm is proposed, allowing for the placement and routing of majority gates and enabling a comparison of design metrics between a layout of a logic network in the MIG representation and its corresponding logic network in the AIG representation.

4.2.1 The proposed signal Distribution Network

The orthogonal algorithm uses 2DDWave clocking, which limits the direction assignment to only two options, *east* and *south*, and can only handle the placement and routing of 2-input logic gates. To introduce "+"-majority gates into the layout, a RES-like clocking scheme is necessary, which includes tiles with three incoming tiles and one outgoing tile. As shown in Figure 2.13(c), such a tile is located at position $(1, 1)$

and is suitable for placing a "+" majority gate, allowing it to be connected with three incoming signals. However, changing the clocking scheme of ortho to RES would be highly inefficient and difficult to implement. This is because if the clocking scheme were completely changed to RES, the algorithm would not be able to utilize every row and column of the clocking. The RES scheme supports signals to flow into the western or northern directions, thus only the first and third rows and the second and fourth columns support eastern and southern signal propagation. This would lead to only these parts of the clocking being utilized for the placement of two-input gates, resulting in approximately double the area usage for only two-input logic gates. An alternative approach to utilizing the RES scheme is to only support it in certain evenly distributed regions, allowing majority gates to be placed in specific, permanently assigned locations. For example, the layout could be divided into 4×4 tile sub-regions, and every fifth sub-region would be RES clocked, while the rest would be occupied with the 2DDWave scheme. On one hand, this approach should not produce as much area overhead since only some regions are inaccessible for two-input logic gates. However, the permanent clocking assignment limits the placement of majority gates to specific spots, leading to large area overhead if a majority gate needs to be placed far away from such a sub-region. For a network consisting mainly of majority gates, this implementation would also waste most of the 2DD-Wave clocked area. Another aspect to consider is the trivial global synchronization constraints within a uniformly 2DDWave clocked layout, which are disrupted by introducing RES clocking within the layout. By introducing irregular clocking such as RES sub-regions, signals can pass different amounts of tiles to reach the same tile, thereby violating the global synchronization constraint. In Figure 4.6, the top four rows are 2DD-Wave clocked and the bottom four rows are RES clocked. In the 2DD-Wave scheme, all three paths start globally synchronized. The two left paths need exactly one clock cycle to reach the majority gate. However, the right path in the RES scheme causes a delay, so that the signal travels two clock cycles before reaching the majority gate, thereby violating the global synchronization constraint.

To overcome these complications, the proposed DN uses a custom clocking scheme only in areas where majority gates are placed, and addresses the global signal synchronization constraint. As a result, the placement and routing of solely two-input gates should not produce any additional area overhead. Figure 4.7 illustrates the proposed majority gate signal DN, which has been integrated into the ortho algorithm. The red marked cells indicate the three inputs for the MGDN. The cells in the middle of the tile are marked because they can be connected from above north or the west, which would result in one normal wire and one bent wire. The output marked blue allows the algorithm to wire it in the *east* or *south* direction, eliminating any limitations. It is also important to note that the input tiles as well as the output tiles have the same clocking number as in a regular 2DDWave scheme, allowing for seamless integration. Although

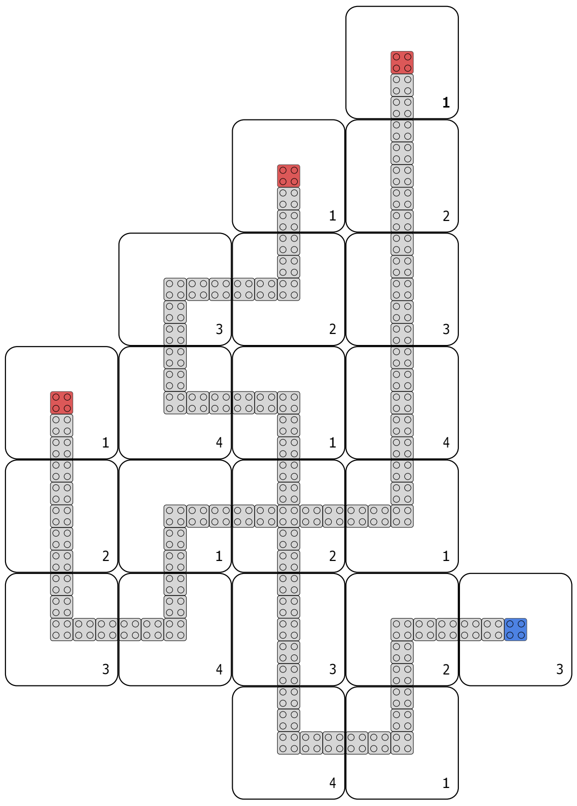


Figure 4.7: Proposed MGDN

the proposed DN does not produce any area overhead for the placement and routing of two-input gates, it can be seen that it does produce excess area for majority gates due to its complex wiring, resulting from the synchronization conditions that had to be considered in its design. In comparison to the implementation of an AIG representation of the majority function designed with the orthogonal algorithm, the placement and routing of this single majority gate already requires more area. However, it should be noted that no wire crossings are used. Assuming that wire crossings have a high cost, the proposed MGDN is considered less costly, although a meaningful cost comparison of the two implementations can only be done under a cost function that is informed by fabrication.

The design constraints used to develop the signal DN are discussed in the following. Firstly, the DN should not contain any wire-crossings as they are considered highly costly. Secondly, the DN must meet the global synchronization constraint. In a 2DDWave clocked layout, every diagonal is synchronous, and every signal wired on the same diagonal passes the same number of tiles following the orthogonal placement and routing. However, when examining the incoming tiles of a three-input tile, it can be seen that only two of the incoming tiles are on the same diagonal, meaning they are globally synchronized, and the third one is shifted by half a clock cycle. This results in the third incoming signal being delayed by half a clock cycle, violating the global synchronization constraint. Additionally, signals must pass a multiple of whole clock cycles in the signal DN in order to support the further use of 2DDWave and the local synchronization constraint. To satisfy this, the initially synchronous signals are first delayed by half a clock signal at the tile where the majority gate is placed, meeting the global synchronization constraint. Then, the output signal of the majority gate is delayed by another half a clock cycle, allowing it to be connected to the regular 2DDWave clocking scheme used in the remaining layout. The delays resulting from the DN lead to a total delay of one whole clock cycle for the signal propagating through the MGDN compared to all other signals in the logic network. Because the delay affects the global synchronization constraint, it has to be considered for each gate connected to the children of a MGDN. In the following, the basic placement and routing of the majority gates DN and the insertion of buffers in order to meet global synchronization is discussed.

4.2.2 Placement and basic routing

The placement and routing of the proposed MGDN are subject to certain constraints. Firstly, the coloring of majority gates must be reviewed, as the logic network now includes three input nodes. However, the coloring algorithm can include auxiliary nodes to resolve coloring conflicts of edges, and therefore, by dividing every edge with

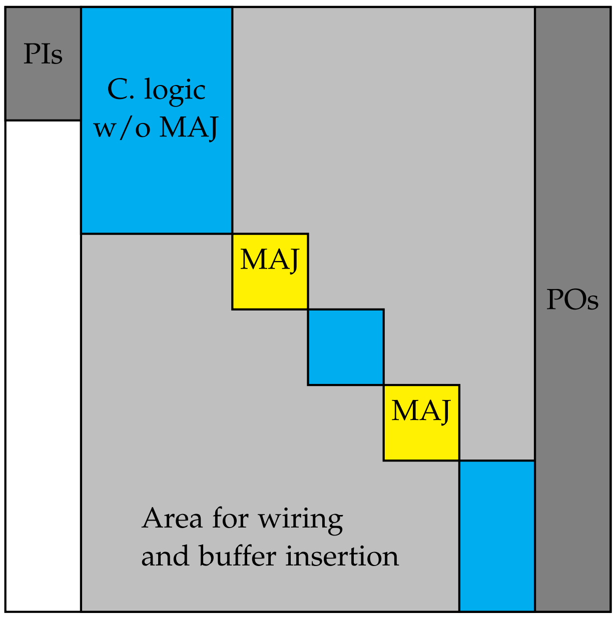


Figure 4.8: Scheme of the P&R using the MGDN

a helping node, it can be seen that a trivial coloring can be found even when including three input nodes in the logic network. Another aspect to consider is the need for a new direction to connect a third signal to the majority gate. However, since the only time such wiring occurs is inside the fixed DN, which can be placed and routed in the usual south-eastern manner, no additional directions need to be included. Additionally, the irregular clocking inside the signal DN must be reviewed. These irregularities do not allow the algorithm to wire connections over the network, requiring a special treatment for the placement of the majority gates. From the algorithm's perspective, a majority gate cannot be placed just *south* or *east* of another gate, as these gates could need wiring through the MGDN. Instead, the algorithm is forced to always assign the MGDN to the *south* and *east* direction to prevent routing conflicts. This means that for majority gates, a naive coloring is always chosen. The major drawback of this is that the area is not used optimally and the layout is extended in two directions as shown in Figure 4.8, compromising the beneficial use of the "+"-majority gate.

4.2.3 Routing with signal synchronization and buffer insertion

The placement and routing using the proposed DN results in a delay of one clock cycles of signals passing through a majority gate. Since the tile-based clocking does not support a speedup of a signal, every other signal which comes into contact with

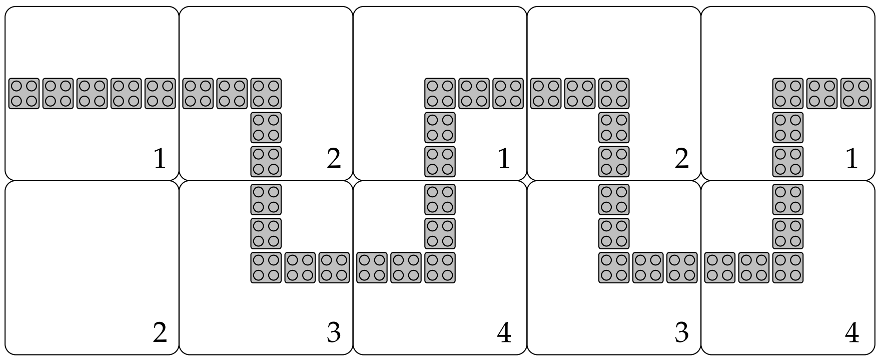


Figure 4.9: Buffer in *east* direction with respective clocking

a delayed signal also has to be delayed in order to meet the global synchronization constraint. Therefore, a function is introduced to compute the delay of signals and allowing signals which are connected together to be synchronized by buffer insertion. For the delay computation, the algorithms views every incoming edge from every node starting at the PO. If an incoming edge is connected to a majority gate, every other incoming edge of the same node gets a delay of one assigned, if this edge is not also connected to a majority gate. In the latter, case all incoming edges of a node are delayed, resulting again in synchronous behavior. The inserted delays then result from the difference of delays of the incoming edges from a node and are realized by inserting wire buffers.

Figure 4.9 depicts a buffer in the *east* direction, which can also be used in the *south* direction by just rotating it by 90 degrees. The snake-shaped structure delays a signal by exactly one clock cycle, but can be extended for any number of clock cycles. This structure is also used in custom placement and routing resulting from the QCA ONE library [46]. As in the MGDN, the buffers support irregular clocking, creating zones through which the algorithm cannot wire. In the case of buffers only one column or row is made impassable, introducing a rewiring for conflicts. Algorithm 3 shows the code snippets which are changed and added to ortho in order to allow the placement and routing of MGDNs. Figure 4.10 shows the placement of a majority gate and one AND gate that needs the incoming signal from the PI to be delayed in order to match the delay of the connected MGDN. The insertion of the buffer happens in *south* direction, while one column is held free in order to resolve any upcoming conflict. The MGDN and its corresponding PIs are marked orange, while the delayed PI and the corresponding buffer are marked green. From this layout, it can already be seen that the implementation of the MGDN brings several complications with it, all resulting in area overhead, which stands in contrast to the area which should be saved by introducing

Algorithm 3 Ortho changes with MGDN

```
⋮
Convert $N$ to a 3-graph by substitution and balance inverters at fan-out nodes, except for majority gates
Compute the delay as majority buffer insertion $buf_{maj}$ for every node and assign it to the incoming signals $\sigma$
Order PI nodes
⋮
for each vertex $v_1, ..., v_i \in N$ with at most three incoming signals $\sigma_1, \sigma_2, \sigma_3$ do
    Rewire incoming signals which are wired on $bl_c$ or $bl_r$
    ⋮
    if vertex $v$ has fan-in of three (is a majority gate) then
        if $d(\sigma_1) = d(\sigma_2) = d(\sigma_3) = south$ then
            Extend $L$ by one row and wire the incoming signal to $(w_p, h-1)$ for every incoming signal
        end if
        Insert majority buffers according to the delay computed in $buf_{maj}$ and safe blocked columns $bl_c$ and rows $bl_r$
        Extend the layout by number of rows (7) and columns (5) of the MGDN and place the DN at $(w-5, h-7)$
        Connect incoming signals west to the inputs of the DN
    else if $d(\sigma_1) = d(\sigma_2) = east$ then
        Insert majority buffers according to the delay computed in $buf_{maj}$ and safe blocked rows $bl_r$
        ⋮
    else if signals are labeled $south$ then
        Insert majority buffers according to the delay computed in $buf_{maj}$ and safe blocked columns $bl_c$
        ⋮
    end if
end for
```

majority gates in the first place.

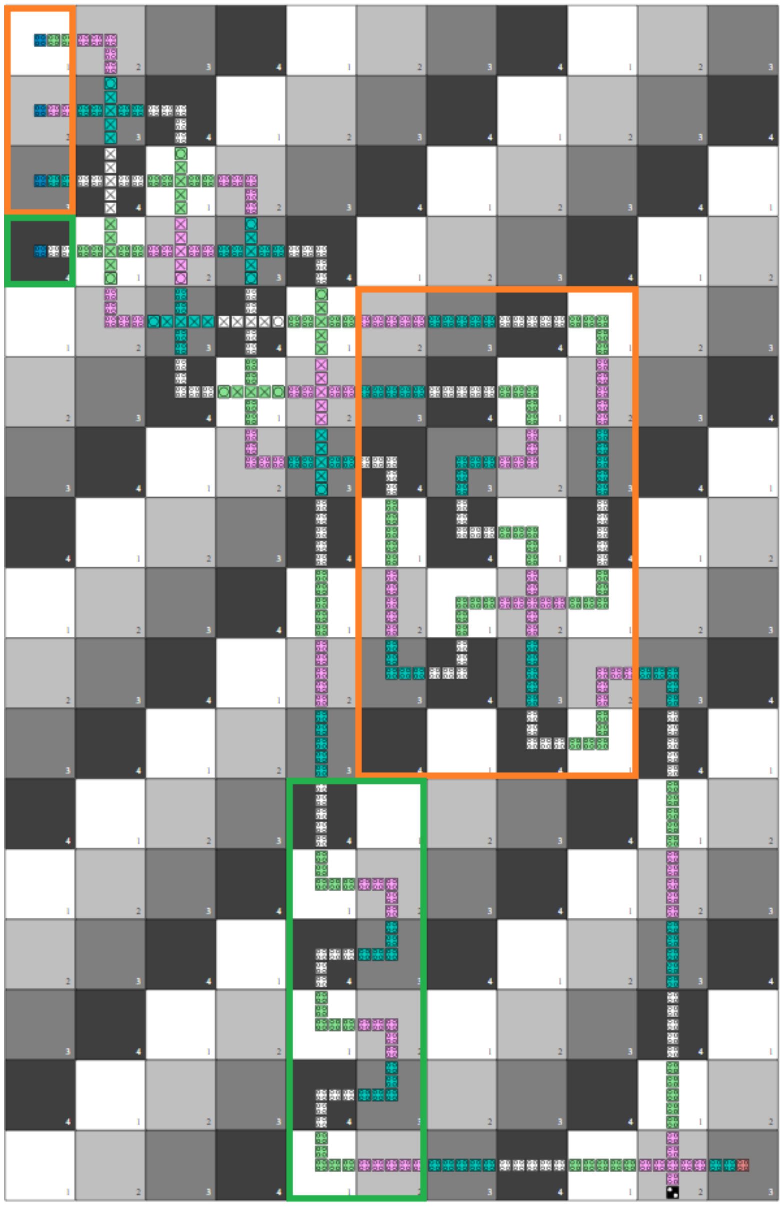


Figure 4.10: Placement and routing of a majority gate in conjunction with a delayed PI

4.3 Sequential Distribution Network (SDN)

In this section, a DN that enables ortho automatic design of sequential circuits is presented. To the authors' knowledge, there is currently no solution for automatically placing and routing sequential circuits in QCA. The existing algorithms for handling sequentiality in QCA, discussed in Chapter 3, simply translate CMOS structures into QCA and rely on an additional external clock signal, which is considered unnatural in the context of QCA's inherent clocking paradigm. The proposed placement and routing method utilizes the concept of signal-delaying wires from [62], and builds upon it to enable automatic placement and routing.

4.3.1 Placement and Routing

Before delving into the algorithm, it is important to first discuss the concept of a storage element in the context of placement and routing, rather than just considering it as a standalone element. From the basic architecture of sequential circuits, we can see that a sequential circuit includes a combinational logic block and a storage element. When translating this directly to QCA, the combinational circuit can be placed as per usual and he storage element can be implemented using a synchronization element(SE). The synchronization element would then stall the signal until new information is entered into the circuit. This has two major drawbacks. First, the proposed SEs implementation in QCA requires more complex clock generators, and it is uncertain if this is possible to implement. Secondly, in contrast to CMOS, where the information can be arbitrarily wired back from the storage to the inputs of the combinational logic, in QCA, wiring back implies the placement of wire segments, each of which delays the information by one clocking zone, acting as a partial FF. When a signal is wired through four adjacent wire gates, a basic FF is formed, since the information is delayed by four clock zones, equivalent to one clock cycle. This delay is the purpose of a clocking element, and the reason why the clocking for the SE is customized. The proposed idea in this work is to use the natural delay that occurs due to sequential wiring to mimic storage elements, reducing the need for customized clocking and allowing wire segments to be summarized as FFs.

Considering the placement and routing of sequential circuits, not only a DN has to be designed but also the logic network has to be expanded regarding storage elements. They are represented in the logic network by registers with their corresponding register input (RI), determining the value, which has to be stored, and its register output (RO), which gives the register value to the combinational logic again, synchronized with the information entered at the PIs. This has to be considered within the clocking of the whole circuit. The registers are implemented into the logic network as follows. RIs

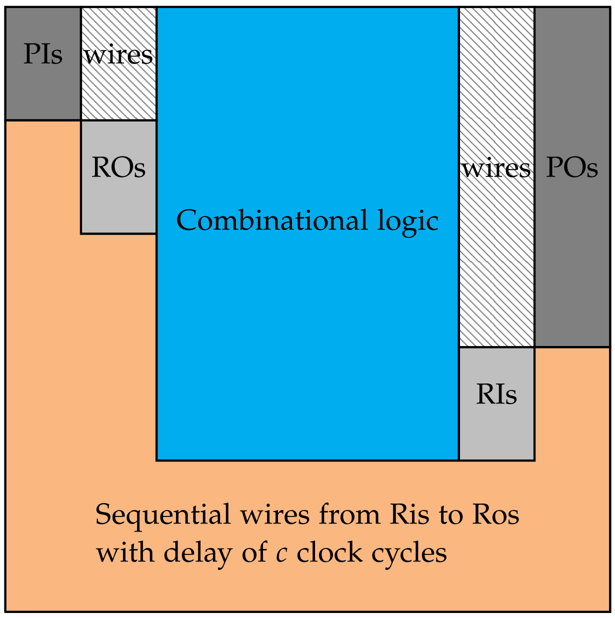

PIs
wires
ROs
Combinational logic
wires
POs
RIs
Sequential wires from Ris to Ros
with delay of c clock cycles

Figure 4.11: Scheme of a sequential circuit layout after placement and routing

are treated similar to POs, therefore they are dangling edges, which point to no node but additionally have a RO assigned. ROs are treated similarly to PIs, being terminal vertices, but always feeding in the data which were given to the corresponding RI, computed from the last inputs of the circuit. Therefore, the logic network extends to $N = (\Lambda, I, RO, \Sigma, O, RI)$.

Also, for placement and routing, the similarities between PIs/ROs and POs/RIs can be exploited. The schematic layout resulting from the described algorithm is shown in Figure 4.11. When first the combinational logic part is placed and routed, treating ROs like PIs and RIs like POs, with the only difference, that PIs and PO are wired to the borders. From this stage, a routing from the RIs to the ROs has to be found, which retains the local and global synchronization constraint. Because every RI has exactly one RO assigned, first of all, the RIs are rewired and sorted in the same order as the ROs. The ordering follows in a way that all RIs are put on a diagonal, and since to this point every gate is clocked uniformly with 2DDWave, the signals are all synchronized. With this starting position, now wires with the same length have to be found between every RI and RO. Since the wires now also have to go in western and northern directions in order to close the loop between the ROs in the upper left corner and the RIs in the down-right corner of the layout, the wiring is not arbitrary. Another big issue is that the clocking cannot be chosen independently for each back-loop because the loops

Algorithm 4 Ortho changes with SDN

```
⋮
if vertex $v$ is PI then
    Extend $L$ by one row
    Place $v$ at position $(0, h-1)$
    Wire the PI to position $(num_{reg} \cdot 2, h-1)$
    if vertex $v$ is colored $south$ then
        Extend $L$ by one column
        Wire the PI to position $(w-1, h-1)$
    end if
else if vertex $v$ is RO then
    Extend $L$ by one row
    Place $v$ at position $(num_{reg} \cdot 2, h-1)$
    if vertex $v$ is colored $south$ then
        Extend $L$ by one column
        Wire the PI to position $(w-1, h-1)$
    end if
    ⋮
end if
Connect the primary outputs to the respective borders
Connect the RIs to the respective borders and connect the RIs to the ROs
return $L$
```

cross each other. Also, the global synchronization constraint has to be viewed again. Considering a PI in a completely combinational circuit being placed in the fifth row of the layout. In this case, the PI is set in a different clock cycle, because its signal is globally delayed by one clock cycle. Until now the assumption was made, that an input network can be used to delay the PI by one clocking cycle to achieve again global synchronization. When an RO is placed in a different clock cycle, this also has to be respected by the wiring of the registers. As already mentioned, the registers do not delay the information by only one clock cycle, but by multiple clock cycles, slowing down the performance of the circuit drastically. This huge delay is due to the fact that ortho lays the combinational logic only in the south-eastern direction, always increasing the distance between PIs/ROs and POs/RIs. Therefore, the SDN always grows with the size of the combinational logic. Figure 4.11 shows, that if the combinational logic grows, also the distance between the RIs and ROs grows, and therefore, the throughput of the circuit is deteriorated.

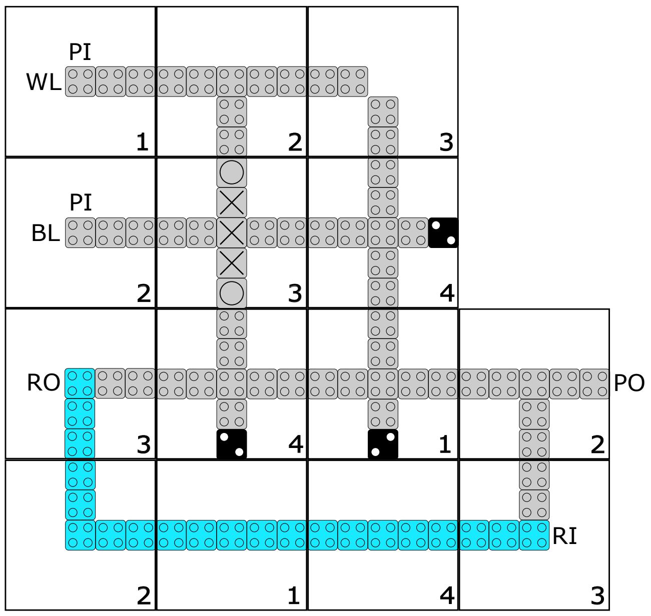


Figure 4.12: QCA RAM cell designed using 2:1 MUX with a SDN

For further research, maybe a folding operation can be found for the ortho algorithm so that the distance between RIs and ROs can be decreased, and therefore, the delay produced by the SDN can be decreased as well.

4.3.2 RAM cell

As previously discussed in Section 3.2.2, a RAM cell can be implemented using wire delays to temporarily store data. Figure 4.12 illustrates a RAM cell, including its WL, BL, and register components. The cell comprises of two main components: (1) a 2:1 MUX and (2) a register (marked turquoise). The register transfers data from the output to one of the inputs of the MUX, which then determines whether new data should be written to the cell or if the existing data should be retained. It is worth noting that the depicted RAM cell was designed manually, however, the proposed SDN is also able to layout a RAM cell, but with the drawback of increased layout area and slower operation.

5 Experimental Evaluation

The proposed Ordering Networks were implemented in C++17 on top of the open-source framework *fiction*, which can be found at https://github.com/marcelwa/fiction. The forked source-code can be found on GitHub with the following link:
https://github.com/hibenj/fiction.

The experimental results were created on a system with a AMD Ryzen 5 PRO 3500U CPU with 8 cores, 16GB RAM and Windows 11 OS.

5.1 Benchmarks

The selected benchmarks aim to compare the Distribution Networks with the state of the art, which is formed by ortho. Hence, the benchmarks [17, 64, 3, 10] ortho has been evaluated on in [71] need to be considered. In case of the ODN all these benchmarks can be evaluated and therefore form a sufficient comparison to the state of the art.

Since only a few of the selected benchmarks contain majority gates, they are not suited to evaluate the properties of MGDNs. Therefore the random network generator in *fiction* is used to create MIGs of different sizes. To recreate these networks the network number is indicating the seed, while aso the number of inputs and the number of gates is given. The MIG benchmark can be placed and routed by the Majority Distribution Network and is converted into an AIG in order to enable ortho to place and route the same logic.

Because the SDN is the first algorithm, which is able to place sequential QCA circuits, no comparison can be made. In this case the *itc99-poli* [61] benchmarks are selected for the evaluation.

5.2 Results

In this section, the evaluation results of the three Distribution Networks are presented and compared against the state-of-the-art with respect to their *key metrics*. The key metrics are those that are expected to be influenced by the Distribution Network, and in addition, common metrics that have been previously investigated in the state-of-the-art are also provided for completeness.

5.2.1 Ordering Distribution Network

The data evaluated for the comparison between the ODN and ortho as state of the art is shown in Table 5.1. The table is ordered as follows: The first column section describes the benchmarks, with their source, name, number of PIs and POs and the number of gates, which are placed on the layout including inverter and fan-out nodes. The second section gives the evaluation results of ortho, including the layout size, represented by the tile number in horizontal (width) and vertical direction (height), the number of wire crossings, number of total wires and the running time. The same metrics are shown for the ODN for comparison. Since the ODN aims to reduce area and wire crossings by ordering the inputs and giving new placement and routing rules, the layout area and number of wire crossings are together with the running time the most important metrics for comparison. Because these functionalities add complexity to the placement and routing algorithm it is expected a slight increase in running time, while the wire crossings and area are reduced. As already stated also an inverter balancing was introduced, which only brings a gate reduction for the *par_check* benchmark. Therefore, the gate number is the same for both algorithms, for *par_check* it has to be considered that it is reduced by one gate. Looking first at the running time of both algorithms, the average is 4% higher for the ODN. The highest running time difference, with plus 75%, is made at the benchmark *c*5315, which is one of the benchmarks with the highest number of gates. Though, the ODN performs very well on this benchmark, reducing wire crossings by 21% and the layout area by 11%. Overall the wire crossings could be reduced by 13%. In case of the *parity* benchmark they could even be reduced by 56%, meaning that wire crossings are not only reduced within the newly utilized input area, but also in the further placement and routing. A negative impact on wire crossings was shown only in four out of 34 cases (12%). Furthermore, the layout area was reduced for all benchmarks with an average of 17%. For smaller benchmarks the area reduction has more percentage improvement, since the area utilization in the PI section has more impact compared to the total size of the layout. For bigger benchmarks this effect is reduced, but with the increase of gates, the area efficient placement and routing of gates can be applied more often. Hence, even for the *c*7552 benchmark with 5654 gates an area reduction of 11% is reached.

	Benchmark			ortho					ODN				
	Name	I/O	|G|	w	h	|WC|	|W|	t in s	w	h	|WC|	|W|	t in s
trindade16 [64]	mux21	3/1	5	6	7	5	24	<1	4	4	1	9	<1
	xor2	2/1	6	5	7	2	17	<1	5	4	1	11	<1
	xnor2	2/1	8	6	8	2	20	<1	6	6	3	18	<1
	par_gen	3/1	14	9	13	6	56	<1	9	10	8	51	<1
	FA	3/2	21	14	16	14	110	<1	13	13	13	99	<1
	par_check	4/1	21	12	19	22	107	<1	11	14	10	154	<1
fontes18 [17]	majority	5/1	21	9	24	22	132	<1	12	16	19	112	<1
	b1_r2	3/4	19	13	17	15	117	<1	11	12	12	95	<1
	1bitAdderAOIG	3/2	21	12	18	14	93	<1	11	15	12	79	<1
	1bitAdderMaj	3/1	36	13	32	28	181	<1	13	29	27	178	<1
	2bitAdderMaj	5/2	59	23	53	62	418	<1	22	51	53	413	<1
	cm82a	5/3	56	22	48	68	475	<1	20	42	54	438	<1
	parity	16/1	133	48	119	164	1867	<1	48	103	72	2212	<1
ISCAS85 [10]	c17	5/2	10	9	12	8	62	<1	7	10	8	44	<1
	c432	36/7	537	187	432	4273	34911	<1	193	419	4063	36732	<1
	c499	41/32	1089	359	841	9422	98988	2.02	328	734	9172	95451	2.02
	c880	60/26	845	292	696	7918	65197	1.41	267	645	8166	65090	1.95
	c1355	41/32	1329	431	1113	6318	103721	2.34	440	1110	5912	104694	3.49
	c1908	33/25	1092	365	819	10300	101085	1.90	342	763	9319	99799	1.92
	c2670	233/63	1840	664	1580	29792	308518	7.19	604	1497	25247	304793	6.42
	c3540	50/22	2748	840	2028	41794	433132	9.71	1949	820	39534	440620	9.39
	c5315	178/123	4615	1616	3436	122373	1551411	34.50	1509	3267	96594	1577735	60.48
	c6288	32/32	6963	1361	5715	31535	629779	24.73	1330	5713	34994	705176	25.86
	c7552	207/107	5654	1710	4318	200597	2312386	56.02	1599	4148	165626	2234848	47.98

	Benchmark			ortho					ODN				
	Name	I/O	|G|	w	h	|WC|	|W|	t in s	w	h	|WC|	|W|	t in s
EPFL [3]	ctrl	7/27	498	185	252	2690	25240	<1	161	344	2497	23703	<1
	int2float	11/7	693	228	495	5533	112860	1.67	222	480	5319	47421	1.05
	router	60/3	659	234	549	7873	51370	1.30	240	502	5699	56821	1.16
	dec	8/256	864	665	472	7161	159743	3.73	410	471	7166	101910	3.04
	cavlc	10/11	1973	578	1428	28690	284337	5.79	568	1393	28325	279467	5.61
	adder	256/129	3055	1027	2670	83063	693203	15.23	899	2668	82809	859067	20.15
	priority	128/8	2761	815	2211	60749	585776	12.17	813	1990	41365	593235	12.30
	i2c	136/127	3507	1329	2576	92862	998017	33.16	1221	2469	84432	1020260	27.53
	bar	135/128	8592	2565	6426	306390	3515968	83.12	2438	6189	283554	3484387	128.79
	max	512/130	7866	2606	6415	544606	5028437	136.49	2506	6158	360324	5289167	142.19
	sin	24/25	14314	4206	10229	506767	7341605	272.73	4183	9776	507042	7139674	182.28

Table 5.1: I/O number of primary inputs/outputs, $|G|$ number of logic network nodes (gates + fan-outs), $w \times h$ aspect ratio given in tiles, $|WC|$ number wire crossings, $|W|$ number of wires, t in s runtime in seconds, OOM maximum RAM reached, —no data available

5.2.2 Majority Gates Distribution Network

For the MGDN the evaluated data is given in Table 5.2. The collected results are again divided in three column sections. The first section describing the benchmark with name, number of PIs and POs and also the number of majority gates in the network. The remaining two sections again include the metrics, number of gates, layout size, represented by the tile number in horizontal (width) and vertical direction (height), number of wire crossings, number of total wires and the running time. Since the benchmarks are given as MIGs, the number of gates differs for the logic network placed and routed with the MGDN and the AIG representation placed and routed with ortho. Due to this property, the MGDN should always have to place fewer gates. As discussed in Section 4.2, because of the placement and routing of the majority gate itself and the additionally required buffer insertion, the majority gates implementation are expected to scale very badly, especially if buffers must be inserted, which is more likely to happen for bigger circuits. On the other hand, since the Majority Distribution Network does not include any wire crossings, a decrease in wire crossings is expected. Even though the number of gates decreases, the complexity of the algorithm is increased dramatically, due to the buffer insertion, leaving the expectation of an increase in running time. Therefore, the key metrics examined for this Distribution Network are the number of placed gates, layout size, number of wire crossings and running time.

Looking at the number of gates placed in these benchmarks, the MIG representation contains on average only 28% of gates compared to their AIG representation. Though an average area increase of 865% and an average running time increase of 457% can be evaluated in the MGDN, making it very costly, while also having bad performance. The only positive property is the decrease of wire crossings by an average of 40%. Depending on a cost metric, which can compare how much area and wires the reduction of one wire crossing is worth, the real impact of this Distribution Network can be evaluated. All the named effects scale with the size of the network. In the smallest network *r*1 all effects are dampened, showing an area increase of 421%, wire crossing reduction of 24%, while the MIG contains only 31% of the gates compared to the AIG network. For both algorithms the benchmark *r*1 finishes in under one second. Looking at the benchmark *r*8 with the most gates, the area and running time increase by 1012% and 666%, while the wire crossings are decreased by 52%. Also the number of gates in the MIG is only 26% the number of gates in the AIG.

Benchmark				ortho						MGDN					
Name	I/O	|N|	|M|	|G|	w	h	|WC|	|W|	t in s	|G|	w	h	|WC|	|W|	t in s
r1	3/2	10	5	84	29	64	92	709	<1	26	107	73	70	1440	<1
r2	4/3	10	7	90	30	72	92	731	<1	28	176	87	78	2377	<1
r3	6/10	40	30	432	138	318	1205	11838	<1	132	904	385	831	34158	1.44
r4	10/27	100	93	1321	390	980	8479	100222	2.42	350	3722	1000	4813	355351	12.91
r5	10/50	200	190	2668	765	1975	31667	371272	14.68	712	8112	1978	16555	1431295	96.20
r6	10/25	90	85	1204	361	890	6803	79105	2.01	310	2744	902	3915	235040	9.71
r7	23/70	277	267	3799	1106	2811	63270	744287	22.72	971	11421	2801	32302	2791464	111.13
r8	16/85	340	324	4638	1321	3436	86195	1050366	29.24	1189	13714	3349	41489	3943616	194.62
r9	16/85	72	61	864	253	643	3907	40424	1.12	245	1815	665	2280	120456	6.82
r10	8/21	88	83	1149	333	855	6861	74636	1.89	306	2706	854	3623	235136	9.74
r11	12/46	149	140	2014	590	1496	19547	221230	5.96	520	6247	1544	9331	857118	43.59

Table 5.2: I/O number of primary inputs/outputs, $|N|$ number of gates before preprocessing, $|M|$ number of majority gates, $|G|$ number of logic network nodes (gates + fan-outs), $w \times h$ aspect ratio given in tiles, $|WC|$ number wire crossings, $|W|$ number of wires, t in s runtime in seconds, OOM maximum RAM reached, —no data available

5.2.3 Sequential Distribution Network

The SDN is the first algorithm, which is able to place sequential logic based on the theory provided in Chapter 4. This means, that in this chapter the first experiments with such an algorithm are made. Therefore, the data in Table 5.3 only contains the evaluation results of the SDN. Again, the rows hold the information for each benchmark problem, which are evaluated in the key metrics number of registers, number of gates, layout area, number of wire crossings, total number of wires and running time. As benchmarks the *itc99-poli* [61] library for sequential circuits is chosen. For benchmarks with more than 245 registers the test system ran out of memory (OOM). The total number of wires is especially important since the implementation of registers is implemented using only wires. For combinational circuits the size of the layout and hence the number of total wires scales with the number of gates in a benchmark. For sequential circuits the number of wires also scales with the number of registers, also directly increasing the layout size and the number of wire crossings. Comparing e.g. benchmark *b*04 with *b*05, it can be seen that *b*04 has about 800 fewer gates, but about 30 registers more than *b*05, which already results in a higher wiring regarding the number of wires and also wire crossings. Therefore the implementation of the SDN can be said to be really costly. The explosion in wiring costs and layout size based on the number of registers is the reason the system gets an OOM issue.

				SDN					
	Name	I/O	|R|	|G|	w	h	|WC|	|W|	t in s
itc99-poli [61]	b01	2/2	5	127	72	107	226	2837	<1
	b02	1/1	4	68	51	56	127	1214	<1
	b03	4/4	30	420	352	376	4421	47514	1.01365
	b04	11/8	66	1866	1032	1469	28841	440650	9.527
	b05	1/36	34	2636	1004	1887	19518	366310	15.2846
	b06	2/6	9	143	102	125	425	4660	<1
	b07	1/8	49	1149	682	941	15163	196220	4.66632
	b08	9/4	21	462	298	387	3428	39367	1.34292
	b09	1/1	28	426	341	365	4619	44597	<1
	b10	11/6	17	549	291	447	4546	47939	1.23536
	b11	7/6	31	1718	683	1303	18781	261924	6.27877
	b12	5/6	121	2854	1706	2313	74904	1195053	42.3646
	b13	10/10	53	878	670	762	14746	174757	10.0356
	b14	32/54	245	24900	8648	18087	876527	26976087	890.069
	b15	36/70	449	-	-	-	-	-	OOM
	b17	37/97	1415	-	-	-	-	-	OOM

Table 5.3: I/O number of primary inputs/outputs, $|R|$ number of registers (D-flipflops), $|G|$ number of logic network nodes (gates + fan-outs), $w \times h$ aspect ratio given in tiles, t in s runtime in seconds, OOM maximum RAM reached, —no data available

6 Conclusion

In this work, the state-of-the-art scalable placement and routing algorithm, ortho, was extended with three signal distribution networks, each adding distinct functionality to the preprocessing and base algorithm.

The ordering distribution network (ODN) aims to reduce wire crossings and layout area by ordering primary inputs and allowing gates to be placed in the input area. The results demonstrate that not only were these goals achieved, but the ordering of primary inputs also affected the placement and routing of the entire logic network resulting in additional benefits in terms of area reduction. Overall, the number of wire crossings was reduced by and average of 17% and the area was reduced by an average of 17%.

The majority distribution network (MGDN) aims to demonstrate the difficulty of applying one of the theoretical advantages of QCA technology, the placement and routing of majority gates. Although the number of gates could be reduced significantly by an average of 72%, the implementation into ortho resulted in higher layout area usage with an average of 865% and wiring effort. This was largely due to the need for buffer insertion, stemming from the 2DDWave scheme used within ortho in combination with the signal delay caused by the placement of majority gates. However it has to be mentioned, that wire crossings were reduced by an average of 40%. To improve the high area usage, a placement of majority gates in the same rows or columns could be considered, at the expense of increased running time. Additionally, using an algorithm that naturally supports clocking, in which majority gates can be implemented, is suggested as a solution to improve the placement and routing of majority gates.

The sequential distribution network (SDN) enhances ortho's ability to design sequential circuits by rethinking the basic theory of sequentiality, storage elements, and clocking for the QCA domain. For the first time sequential circuits could be placed with an automatic placement and routing algorithm. Wire registers were proposed and used in the placement and routing, and this has to be considered for the Bennett clocking and slows down the circuit, depending on the size of the layout's combinational part. The concepts of wire registers were powerful enough to implement a RAM cell based on a 2:1 MUX and are suggested for further research in implementing sequential behavior into QCA circuits.

List of Figures

List of Tables

Bibliography

[1] E. Ábrahám and G. Kremer. “Smt solving for arithmetic theories: Theory and tool support.” In: *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE. 2017.

[2] F. Ahmad. “An optimal design of QCA based multiplexer/demultiplexer and its efficient digital logic realization.” In: *Microprocessors and Microsystems* 56 (2018).

[3] L. Amarú, P.-E. Gaillardon, and G. De Micheli. “The EPFL combinational benchmark suite.” In: *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*. 2015.

[4] T. Ball, A. Podelski, and S. K. Rajamani. “Boolean and Cartesian abstraction for model checking C programs.” In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2001.

[5] C. Bandyopadhyay, R. Das, A. Chattopadhyay, and H. Rahaman. “Design and synthesis of improved reversible circuits using AIG-and MIG-based graph data structures.” In: *IET Computers & Digital Techniques* 13.1 (2019).

[6] Z. Beiki and A. Shahidinejad. “An Introduction to Quantum Cellular Automata Technology and Its Defects.” In: *Reviews in Theoretical Science* 2 (Dec. 2014).

[7] D. Bhowmik, A. K. Pramanik, J. Pal, P. Sen, A. R. Singh, A. K. Saha, and B. Sen. “Regular clocking-based Automated Cell Placement technique in QCA targeting sequential circuit.” In: *Computers & Electrical Engineering* 98 (2022).

[8] T. Biedl and G. Kant. “A better heuristic for orthogonal graph drawings.” In: *Computational Geometry* 9.3 (1998).

[9] E. Blair and C. Lent. “An architecture for molecular computing using quantum-dot cellular automata.” In: *Third IEEE Conference on Nanotechnology, 2003. IEEE-NANO 2003.* Vol. 1. IEEE. 2003.

[10] F. Brglez and H. Fujiwara. “A neutral netlist of 10 combinational benchmark circuits and a target translator.” In: *Fortran. ISCAS’85*. 1985.

[11] R. E. Bryant. “Graph-based algorithms for boolean function manipulation.” In: *Computers, IEEE Transactions on* 100.8 (1986).

[12] M. Bubna, S. Roy, N. Shenoy, and S. Mazumdar. "A layout-aware physical design method for constructing feasible QCA circuits." In: *Proceedings of the 18th ACM Great Lakes symposium on VLSI*. 2008.

[13] C. A. T. Campos, A. L. Marciano, O. P. V. Neto, and F. S. Torres. "USE: a universal, scalable, and efficient clocking scheme for QCA." In: *IEEE Transactions on computer-aided design of integrated circuits and systems* 35.3 (2015).

[14] J. Chaharlang and M. Mosleh. "An overview on RAM memories in QCA technology." In: *Majlesi Journal of Electrical Engineering* 11.2 (2017).

[15] W.-J. Chung, B. Smith, and S. K. Lim. "Node duplication and routing algorithms for quantum-dot cellular automata circuits." In: *IEE Proceedings-Circuits, Devices and Systems* 153.5 (2006).

[16] M. Eiglsperger, S. P. Fekete, and G. W. Klau. "Orthogonal graph drawing." In: *Drawing Graphs*. Springer, 2001.

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

[18] R. E. Formigoni, R. S. Ferreira, and J. A. M. Nacif. "Ropper: A placement and routing framework for field-coupled nanotechnologies." In: *Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE. 2019.

[19] A. Gin, P. D. Tougaw, and S. Williams. "An alternative geometry for quantum-dot cellular automata." In: *Journal of Applied Physics* 85.12 (1999).

[20] M. Goswami, A. Mondal, M. H. Mahalat, B. Sen, and B. K. Sikdar. "An efficient clocking scheme for quantum-dot cellular automata." In: *International Journal of Electronics Letters* 8.1 (2020).

[21] A. Grabowski. "Mechanizing complemented lattices within Mizar type system." In: *Journal of Automated Reasoning* 55.3 (2015).

[22] P. Halmos and S. Givant. *Introduction to Boolean algebras*. Springer, 2009.

[23] C. Hawkins, J. Segura, and P. Zarkesh-Ha. *CMOS Digital Integrated Circuits: A First Course*. Materials, Circuits and Devices. Institution of Engineering and Technology, 2012.

[24] E. V. Huntington. "Boolean Algebra. A Correction." In: *Transactions of the American Mathematical Society* 35 (1933).

[25] E. V. Huntington. "A New Set of Independent Postulates for the Algebra of Logic with Special Reference to Whitehead and Russell's Principia Mathematica*." In: *Proceedings of the National Academy of Sciences* 18.2 (1932). eprint: `https://www.pnas.org/doi/pdf/10.1073/pnas.18.2.179`.

[26] P. T. Johnstone. "Conditions related to De Morgan's law." In: *Applications of sheaves* (1979).

[27] J. S. D. Júnior, L. S. da Rosa Júnior, and F. de Souza Marques. "Migortho: A Design Automation Flow for QCA Circuits." In: *IEEE Design & Test* 39.2 (2021).

[28] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu. *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.

[29] R. W. Keyes and R. Landauer. "Minimal energy dissipation in logic." In: *IBM Journal of Research and Development* 14.2 (1970).

[30] D. Kumar and D. Mitra. "A systematic approach towards fault-tolerant design of QCA circuits." In: *Analog Integrated Circuits and Signal Processing* 98 (Mar. 2019).

[31] R. Landauer. "Irreversibility and heat generation in the computing process." In: *IBM journal of research and development* 5.3 (1961).

[32] C. S. Lent, M. Liu, and Y. Lu. "Bennett clocking of quantum-dot cellular automata and the limits to binary logic scaling." In: *Nanotechnology* 17.16 (2006).

[33] C. S. Lent and P. D. Tougaw. "A device architecture for computing with quantum dots." In: *Proceedings of the IEEE* 85.4 (1997).

[34] C. S. Lent, P. D. Tougaw, and W. Porod. "Quantum cellular automata: the physics of computing with arrays of quantum dot molecules." In: *Proceedings Workshop on Physics and Computation*. IEEE. 1994.

[35] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein. "Quantum cellular automata." In: *Nanotechnology* 4.1 (1993).

[36] L. A. Lim, A. Ghazali, S. C. T. Yan, and C. C. Fat. "Sequential circuit design using Quantum-dot Cellular Automata (QCA)." In: *International Conference on Circuits and Systems (ICCAS)*. IEEE. 2012.

[37] M. Mahdavi, M. A. Amiri, S. Mirzakuchaki, and M. N. Moghaddasi. "Single Electron Fault in QCA Inverter Gate." In: *International Conference on MEMS NANO, and Smart Systems*. 2009.

[38] A. H. Majeed, E. Alkaldy, M. S. Zainal, K. Navi, and D. Nor. "Optimal design of RAM cell using novel multiplexer in QCA technology." In: *Circuit world* (2019).

[39] A. H. Majeed, M. S. Zainal, and E. Alkaldy. "Quantum-dot Cellular Automata." In: *International Journal of Integrated Engineering* 11.8 (2019).

[40] M. Mohammadi, M. Mohammadi, and S. Gorgin. "An efficient design of full adder in quantum-dot cellular automata (QCA) technology." In: *Microelectronics journal* 50 (2016).

[41] R. S. Mohammed Alharbi Gerard Edwards. "Novel Ultra-Energy-Efficient Reversible Designs of Sequential Logic Quantum-Dot Cellular Automata Flip-Flop Circuits." In: (2022).

[42] A Orlov, A Imre, G Csaba, L Ji, W Porod, and G. Bernstein. "Magnetic quantum-dot cellular automata: Recent developments and prospects." In: *Journal of Nanoelectronics and Optoelectronics* 3.1 (2008).

[43] F. Peng, Y. Zhang, R. Kuang, and G. Xie. "Spars: A Full Flow Quantum-Dot Cellular Automata Circuit Design Tool." In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 68.4 (2020).

[44] S. Perri and P. Corsonello. "New methodology for the design of efficient binary addition circuits in QCA." In: *IEEE Transactions on Nanotechnology* 11.6 (2012).

[45] F. Pigorsch, C. Scholl, and S. Disch. "Advanced unbounded model checking based on AIGs, BDD sweeping, and quantifier scheduling." In: *Formal Methods in Computer Aided Design*. IEEE. 2006.

[46] D. A. Reis, C. A. T. Campos, T. R. B. S. Soares, O. P. V. Neto, and F. S. Torres. "A Methodology for Standard Cell Design for QCA." In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2016.

[47] J. I. Reshi, M. T. Banday, and F. A. Khanday. "Sequential circuit design using quantum dot cellular automata (QCA)." In: *Symposium on Computers, Communication and Electronic Engineering*. 2015.

[48] J. Retallick and K. Walus. "Population congestion in 3-state quantum-dot cellular automata." In: *Journal of Applied Physics* 127.24 (2020).

[49] N. Safoev and J.-C. Jeon. "Design of high-performance QCA incrementer/decrementer circuit based on adder/subtractor methodology." In: *Microprocessors and Microsystems* 72 (2020).

[50] T. N. Sasamal, A. K. Singh, and A. Mohan. "Quantum-Dot Cellular Automata Based Digital Logic Circuits: A Design Perspective." In: *Studies in Computational Intelligence*. 2020.

[51] R. R. Schaller. "Moore's law: past, present and future." In: *IEEE spectrum* 34.6 (1997).

[52] G. Schedelbeck, W. Wegscheider, M. Bichler, and G. Abstreiter. "Coupled quantum dots fabricated by cleaved edge overgrowth: From artificial atoms to molecules." In: *Science* 278.5344 (1997).

[53] C. Scholl and P. Molitor. *Communication based FPGA synthesis for multi-output Boolean functions*. IEEE, 1995.

[54] G. Schulhof, K. Walus, and G. A. Jullien. "Simulation of Random Cell Displacements in QCA." In: *J. Emerg. Technol. Comput. Syst.* 3.1 (2007).

[55] B. Sen, A. Nag, A. De, and B. K. Sikdar. "Towards the hierarchical design of multilayer QCA logic circuit." In: *Journal of Computational Science* 11 (2015).

[56] R. Singh and M. K. Pandey. "Analysis and implementation of reversible dual edge triggered d flip flop using quantum dot cellular automata." In: *Int. J. Innov. Comput. Inf. Control* 14.1 (2018).

[57] R. Singh and D. K. Sharma. "Design of efficient multilayer RAM cell in QCA framework." In: *Circuit World* (2020).

[58] M. Taucer, F. Karim, K. Walus, and R. A. Wolkow. "Consequences of many-cell correlations in clocked quantum-dot cellular automata." In: *IEEE Transactions on Nanotechnology* 14.4 (2015).

[59] T. Teodósio and L. Sousa. "QCA-LG: A tool for the automatic layout generation of QCA combinational circuits." In: *Norchip 2007*. IEEE. 2007.

[60] H. Thapliyal and N. Ranganathan. "Reversible logic-based concurrently testable latches for molecular QCA." In: *IEEE transactions on nanotechnology* 9.1 (2009).

[61] C. G. at Politecnico di Torino. *PoliTo ITC99*. 1999. URL: `https://github.com/squillero/itc99-poli` (visited on 06/26/2018).

[62] F. S. Torres, M. Walter, R. Wille, D. Große, and R. Drechsler. "Synchronization of clocked field-coupled circuits." In: *International Conference on Nanotechnology (IEEE-NANO)*. IEEE. 2018.

[63] G. Toth and C. S. Lent. "Quasiadiabatic switching for metal-island quantum-dot cellular automata." In: *Journal of Applied Physics* 85.5 (1999).

[64] A. Trindade, R. Ferreira, J. A. M. Nacif, D. Sales, and O. P. V. Neto. "A placement and routing algorithm for quantum-dot cellular automata." In: *Symposium on integrated circuits and systems design (SBCCI)*. IEEE. 2016.

[65] G. Turvani, A Tohti, M. Bollo, F. Riente, M. Vacca, M. Graziano, and M. Zamboni. "Physical design and testing of nano magnetic architectures." In: *IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE. 2014.

[66] V. Vankamamidi, M. Ottavi, and F. Lombardi. "Two-dimensional schemes for clocking/timing of QCA circuits." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.1 (2007).

[67] M. Walter and R. Drechsler. "Design Automation for Field-Coupled Nanotechnologies." In: July 2020.

[68] M. Walter, W. Haaswijk, R. Wille, F. S. Torres, and R. Drechsler. "One-pass synthesis for field-coupled nanocomputing technologies." In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2021.

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

[70] M. Walter, R. Wille, D. Große, F. S. Torres, and R. Drechsler. "Placement and Routing for Tile-Based Field-Coupled Nanocomputing Circuits Is NP-Complete (Research Note)." In: *J. Emerg. Technol. Comput. Syst.* 15.3 (2019).

[71] M. Walter, R. Wille, F. S. Torres, D. Große, and R. Drechsler. "Scalable design for field-coupled nanocomputing circuits." In: *Asia and South Pacific Design Automation Conference*. 2019.

[72] K Walus, A Vetteth, G. Jullien, and V. Dimitrov. "RAM design using quantum-dot cellular automata." In: *Nanotechnology conference*. Vol. 2. 2003.

[73] M. Wilson, K. Kannangara, G. Smith, M. Simmons, and B. Raguse. "Nanotechnology: basic science and emerging technologies." In: (2002).

[74] L. Zhang. "Solving QBF with combined conjunctive and disjunctive normal form." In: *Proceedings of the national conference on artificial intelligence*. Vol. 21. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2006.