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

Signalverteilungs-Netzwerke in automatisierter QCA Standard Zellen Plazierung und Verdrahtung

Author:	Benjamin Hien
Supervisor:	Prof. Dr. Robert Wille
Advisor:	Dr. Marcel Walter
Submission Date:	08.02.2023

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

Munich, 08.02.2023 Benjamin Hien

Acknowledgments

Abstract

New technologies to compete with CMOS, one of them QCA
Placement and Routing as key to producibility.
Challenges of Placement and Routing in previous algorithms.
Goals of this work:
I minimizing area/tiles,
II making it possible to place majority-gates (which is a promising aspect of QCA),
III making it possible to P&R sequential circuits
This is done by introducing several signal distribution networks
Results are compared with already existing algorithms...

Contents

1 Introduction

1.1 Motivation

About the technology, why its promising and important.
Lack of automated algorithms for P&R
P&R as sign of producibility
Why the distribution networks are able to make QCA better producible / cheaper

1.2 Objective

The thesis is divided in ...

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 three fields forming this basis are the representation of Logic Circuits, QCA technology and the placement and routing problem.

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. 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 [20, 19], 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 $+ : B_{abc} \times B_{abc} \rightarrow B_{abc}$ and the unary function $' : B_{abc} \rightarrow B_{abc}$.

$$a + b = b + a$$
$$(a + b) + c = a + (b + c)$$
$$(a' + b')' + (a' + b)' = a$$

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:

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

Even though the two operands disjunction and negation function are powerful enough to form a Boolean algebra, the most common definition also uses the conjunction function $\cdot : B_{abc} \times B_{abc} \to B_{abc}$ in order to form shorter and more readable logic terms. The most common Boolean algebra $\mathbb{B}$ is defined by the tuple $(B_{abc}, \vee, \wedge, \neg)$, where $\vee$ and $\wedge$ are other denotations for binary operands for disjunction $+$ and conjunction $\cdot$ in $\mathbb{B}$. The third function $\neg$ describes the unary negation function, which was previously denoted as $'$. The set B_{abc} contains exactly two distinct elements $\{0, 1\}$, with $u = 1$ and $\neg u = 0$ respectively [17].

Since this definition is restricting the use of only the three Boolean functions $(\vee, \wedge, \neg)$, we want to extend it by the following definition [42]:

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 given in 2.1.3, which is the most important Boolean function used in 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. A literal is either an atom a (positive literal) or the negation of an atom $\neg a$ (negative literal).

Definition 2.1.5. 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 [58]:

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

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* [21]

Definition 2.1.6. 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*.

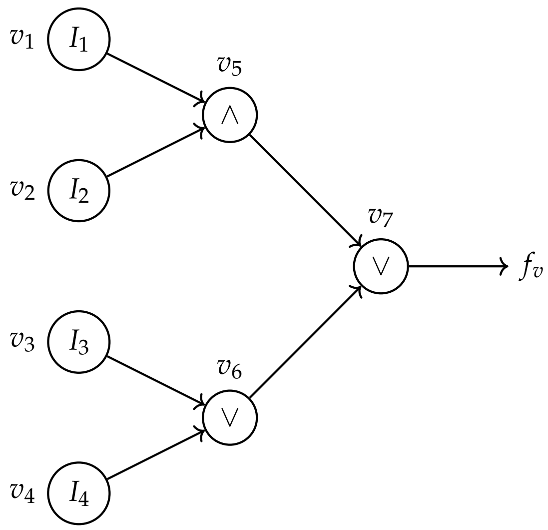
2.1.2 Logic Networks

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 [36]. Logic networks overcome these restrictions being *non-canonical*, meaning that a given function can be represented by different logic networks. The following definition for Logic Networks is derived from [7]:

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 children. A vertex connected by an outgoing edge $e_{out} \in E$ is called parent. V contains two types of vertices. A *non-terminal* vertex v has as attributes an argument index $index(v) \in \{1, ..., n\}$, and l children $child_1(v), ..., child_l(v) \in V$. A *terminal* vertex v has as attribute a value $value(v) \in \{0, 1\}$.

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

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



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

Figure 2.1: Binary Logic Network

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 $f_v(v_i) = x_i(f_{child_1(v)}(v_{i-1}), ..., f_{child_l(v)}(v_{i-n}))$.

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 negotiation 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 [53]. Attempts to create canonical logic networks, seem to evade this problem, but include *co*$\mathcal{NP}$-complete problems in itself [7]. Nevertheless, logic networks have proven to be very useful in transforming logic circuits into gate representations called *logic synthesis*. Due to the complexity of the representations algorithms commonly used for logic synthesis are based on approximate solutions.

Adapting the names used in the literature, 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 child v_i and a parent 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 subsection 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) allowing only conjunction and negation. Another binary logic network, which is used in the QCA domain, is the *Majority-Inverter Graphs* (MIGs) 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 *XOR-AND-Inverter Graph* (XAG) with the parity function, conjunction, and negation functions, respectively [53].
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 [40], 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 [28, 50, 57]. To tackle these challenges among others the International Roadmap for Devices and Systems (IRDS, former ITRS), provides a platform proposing solutions within the semiconductor domain, e.g. new materials and multi-core architectures. But new technologies are also being researched, including quantum computing and the domain of *Field-Coupled Nanocomputing* (FCN). 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 [33]. Hence, QCA tackles exactly the main issues faced with 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 [32]. Because this work focuses on the design of QCA circuits, it mainly views circuit parameters such as area, complexity, and clock delays [2]. In order to allow an analysis under these parameters, first the QCA technology and the resulting design constraints have to be understood. Therefore, subesection 2.2.1 introduces QCA cells as building blocks for QCA gates, which are discussed in subesection 2.2.2. After understanding the clocking in subsection 2.2.3, the constraints for the placement and routing problem can be formulated subsequently.

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 [4]. In order to achieve this property a nanosized structure is needed, which is capable of trapping electrons in a certain position. For this purpose so-called *quantum-dots* (QDs) are utilized. A QD consists of several to one hundred atoms of a semiconductor, and therefore quantum mechanics applies for their electrical properties. 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

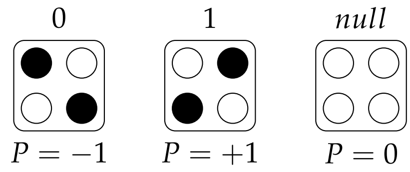


Figure 2.2: QCA-Cell sates

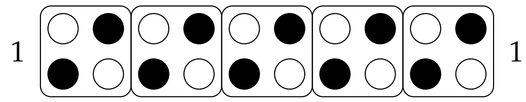
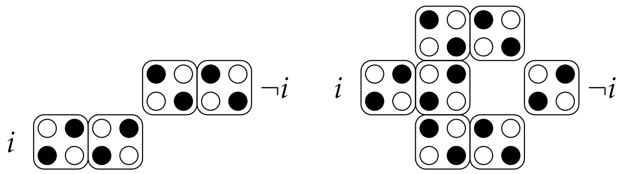


Figure 2.3: Adjacent QCA-cells forming a wire segment

artificial molecule [41]. Figure 2.2 shows three such QCA cells, where the four circles at the corners of each QCA cell show the QDs or rather their quantum barriers, which are capable of trapping 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 [39, 26, 27].
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 [39]. 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 [26]. 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.



(a) 45° Inverter (b) Standard Inverter

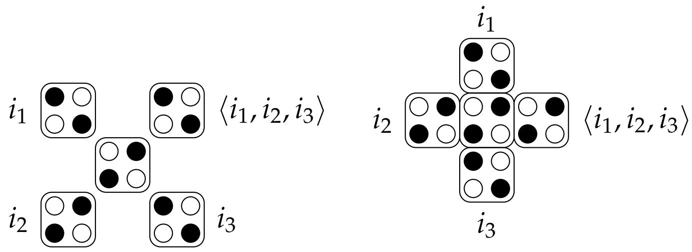
Figure 2.4: Different QCA Inverter representations

2.2.2 Gates

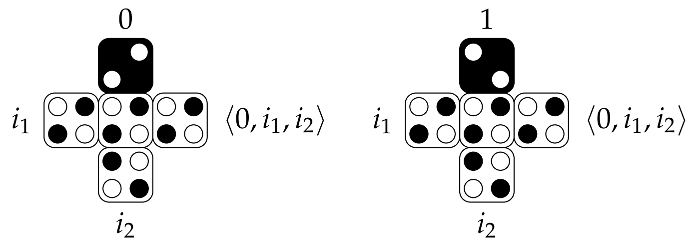
In this subsection, QCA cells are combined to form different logic gates, which form the gate library used later for the design of QCA logical circuits. Some of these gates are inherited from the QCA ONE library [37], which is already used fully [35] or partially [55, 13] as a basis for some works. The QCA ONE library proposes gates formed by one tile as well as gates formed by multiple tiles. A major drawback of this library is the prerequisite of a clocking scheme (USE) in order to form multiple tile gates. This restricts the underlying placement and routing algorithm in the clocking domain. Also manual changes of the standard cells clock zones, size or positioning is not allowed [37], imposing the designer with even more restrictions. For this work, the standard cell library should contain only gates that occupy one tile. Every other logic function is composed out of these standard cells. The tiles used are of the dimension 5×5 cells, which means that all standard gates are reduced to this area.

Until now, inverters and majority gates have been 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 [39]. In order to get a more robust gate regarding disturbance, the C-shaped inverter shown in figure 2.4(b) is introduced [37]. This gate is used as standard in many libraries and works [35, 13, 29, 9], but it should be mentioned that this implementation is really prone to complex single-electron faults [30] and even common displacement faults [43], 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 most important gate. Definition 2.1.3 suggests that the implementation of this function in CMOS technology requires multiple AND and OR gates to be



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

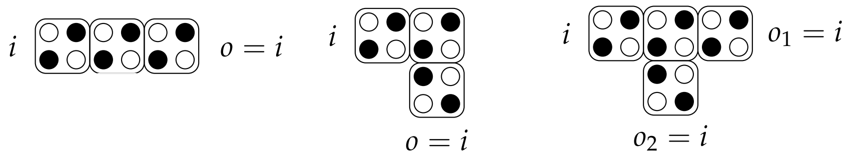


(c) Standard AND Gate (d) Standard OR Gate

Figure 2.5: The QCA Majority gate

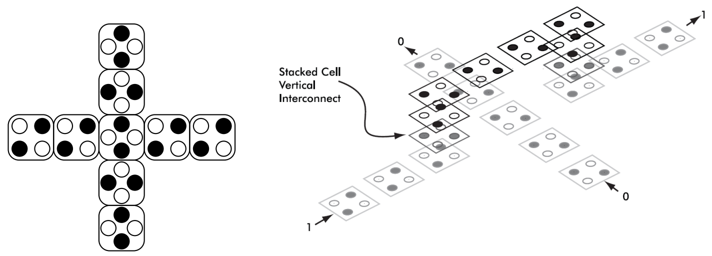
placed and routed. In QCA technology on the other hand a majority function can be represented by exactly one gate, making it one of the major advantages over CMOS technology. There are two main implementations of the majority gate. The rotated majority gate in Figure 2.5(a), which is used in QCA ONE, 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 [23]. 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 work. 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).

Different than in CMOS technology in QCA, wires must also be introduced as gates.



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

Figure 2.6: QCA wires

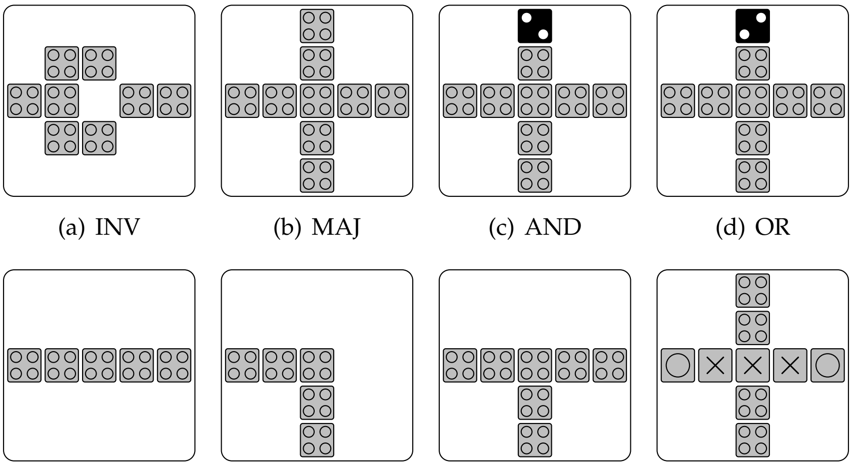


(a) Coplanar wire crossing (b) Multi-layer wire crossing [43]

Figure 2.7: Different QCA wire crossing implementations

As already seen in figure 2.3 a QCA wire also consists of QCA cells and therefore forms a gate. Since wires do not add functionality to the logic, they are viewed as logic gates representing the unity function. This property has the huge drawback that the cost of wires is comparable with other logic gates, which is being used as one major cost metric for the circuit design. Until now only straight planar wires have been introduced. From the implementation of the majority gate it can already seen that data is not only transferred in the x-Dimension but also in 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. In order to allow information to change its propagation direction from the x-direction to the y-direction and vice versa, also bent wires have to be introduced. They are depicted in Figure 2.6(b) and show a 90-degree bend. Given that 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. Also, since all gates introduced so far have a fan-out of one, we need a fan-out node to duplicate signals. This is done 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



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

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

Figure 2.8: QCA Standard Library

string by 45° the rotated cells do not have crosstalk with nonrotated cells [43] as shown in figure 2.7(a). This solution is very handy because it supports the planar structure of the circuit and is therefore called *coplanar wire crossing*. Further the possibility of multi-layer QCA has been investigated [15] and found especially useful in the case of wire crossings. To use this, one wire string is raised to an additional higher layer, which is connected with a vertical interconnect as in figure 2.7(b). The signal transmission in the vertical-stacked cells works just as in the horizontal direction. To impede any crosstalk between the wire strings, two intermediate layers of cells are used in the vertical direction. Theoretically, the top layer cannot only be used as wire, but since the signal distribution works just 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. This makes the horizontal interconnect very sensitive to crosstalk and therefore highly prone to cell displacements. Multilayer circuits, on the other hand, show high robustness and therefore are used as standard in this library. In the gate 5×5 representation of the multilayer interconnect, the top wire string is described with a $\times$, while the vertical layers are described with a circle. Although it has to be mentioned that the complex structure of the multilayer wire crossing yields high costs and is therefore tried to be avoided in the design of QCA circuits. In Figure 2.8 all gates used in this work are summarized.

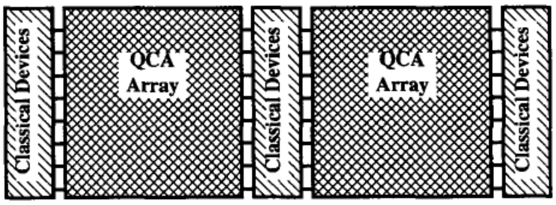


Figure 2.9: Schematic of a combined QCA and CMOS system [27].

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 repulsion and changes its polarization accordingly. Looking 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 driver 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*. When a cell has no adjacent cell in the distribution direction, 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 rather sloshes around, showing a quite unpredictable behavior. 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, is introduced. Another reason for a clocking is, that for the described straight forward process called abrupt switching 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 clocking zone receives an external signal, a clock, assigned. 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 clocking zone, when applying a new input. This means that the electrons are not trapped and can

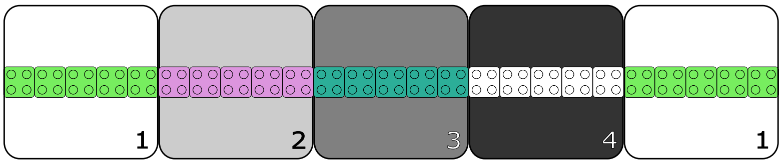
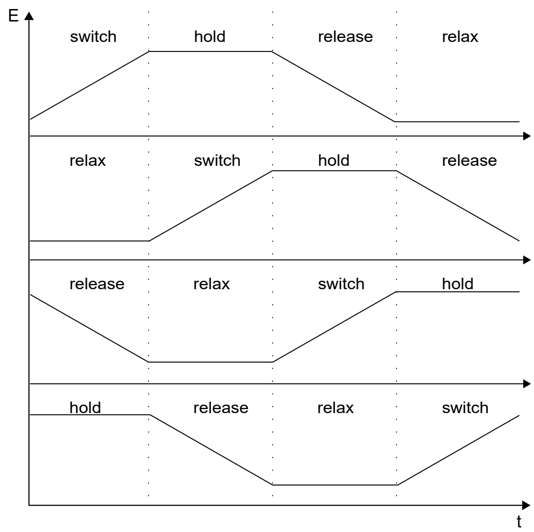


Figure 2.10: QCA wire devised into the four clock zones according to Bennet clocking

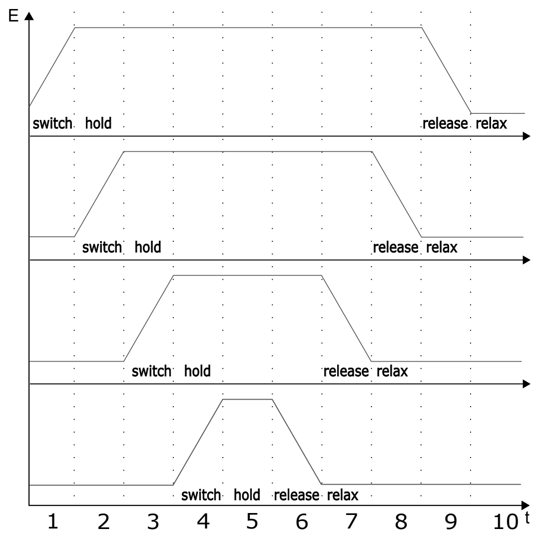
move freely following Coulomb repulsion, therefore taking over the polarization of the input cell. When all cells in the region are stable, the barriers are raised again, localizing the electrons in the cells, which now have the desired polarization. Meanwhile the barriers of the subsequent clocking zone are lowered simultaneously, the previous sub-region acts as fixed input and again the polarization is taken over. In this way, the information gradually propagates through the whole circuit [26]. 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 a equivalent effect as in the former approach [53].

A wire, divided in such clock zones is shown in 2.10. The colors of the zones and cells, as well as the zone number, show redundant information about the type of clocking zone. They differ in the external applied electrical field and therefore the energy of the cells. In QCA the clocking is divided into the 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 $\pi/2$, forming a 2π clock cycle. In the switch phase, cells start to get polarized, dependent 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 [39]. After one clocking cycle, the next clocking cycle starts with the same order of states also notated with numbers $i = \{1, 2, 3, 4\}$. Hence, for consecutive clocking numbers holds ($i_{next} = i_{previous} + 1 \mod clk$). The scheme of such a pipeline as clocking is depicted exemplaryly in Figure 2.11(a).

The described clocking is named *Landauer clocking*. The inventor Rolf Landauer himself pointed out the vast power dissipation of this clocking mechanism. The main cause for the high power dissipation is the *erase* function, which happens because in the Landauer clocking the release state directly follows on the hold phase, irreversibly erasing the information and therefore transforming it into heat [24]. To tackle this problem in the QCA domain, Landauer pointed out, that the erase function has to be eliminated from the clocking. He argues that every erased bit dissipates at least $k_B T \ln(2)$ in heat dissipation [22]. Exemplary if a QCA-cell has size $1nm \times 1nm$ and



(a) Landauer clocking



(b) Bennet clocking

Figure 2.11: QCA clocking pipeline

operating frequency of $100GHz$, the corresponding density of devices results to 10^{14} cm^{-2}. Further a dissipation of $0.1eV$ every clock cycle is assumed, resulting in a total power dissipation of 160 $kW\ cm^{-2}$. This directly yields the statement that a device operating with this clocking would be inoperable (it would evaporate due to heat) [25]. The *Bennet clocking* tackles exactly this problem by altering the timing of the clocking 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 was 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. The effective clock rate is at least halved due to the additional backwards propagation and since only one signal vector can be transmitted through the system, the pipeline capabilities are reduced [25]. The resulting clocking scheme for Bennet clocking is shown in figure 2.11(b).

In order to apply Bennet clocking, it was already stated that the QCA-array has to be divided into clock zones. 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 an uniform fabrication in order to fabricate

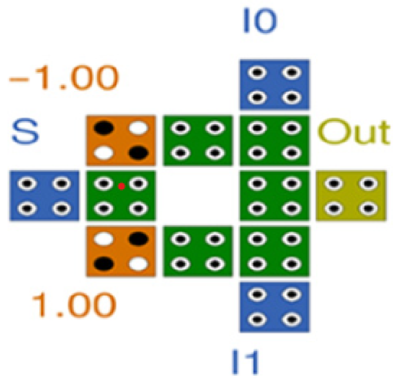
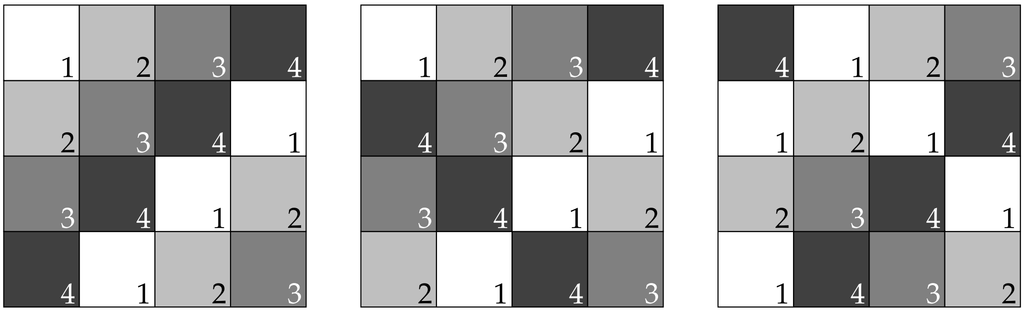


Figure 2.12: Cell based layout of a 2:1 mux [31]



(a) 2DDWave (b) USE (c) RES

Figure 2.13: Different clocking Schemes in QCA

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, in order to provide a clocking signal to the single-cell region. Since this is also not feasible, this design is obsolete. An example of a cell-based clocking design of a 2:1 mux can be seen in 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 following an argument against cell-based clocking [46].

The tile-based clocking leads to several proposals of clocking-schemes, which give a certain distribution of clock zones. Since they follow an uniform pattern they can be extended easily for every size of the circuit. In figure 2.13 three clocking schemes are shown, each of them 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 [52]. This simplicity allows no back propagation, prohibiting the placement and routing of sequential circuits. Also, it restricts gates in the scheme to have a maximum input size of two. The USE scheme,shown in figure 2.13(b), tackles the first problem by introducing clocking loops into the scheme, giving the possibility to place sequential circuits [9]. To tackle the second problem, the RES scheme, shown in figure 2.13(c), gives the opportunity to place gates of input size three. Since one tile is restricted to four adjacent tiles, of which one has to 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 [16]. In QCA technology, they can be represented by only one tile, making them a huge advantage over CMOS technology. This is further evaluated in the next subsection on gates.

2.3 Placement and routing problem

In simple words the placement and routing (P&R) problem can be formulated as the placement of logic gates, which are represented by vertices in a logic network and the routing between gates via wires, which are represented as edges in a logic network, in a way that the Boolean function of the logic network is retained in the designed circuit. Because each gate with its position in the layout is strongly dependent on the clocking inside the QCA domain additionally the circuit has to follow the local and global timing constraints in order to function correctly. In this section, the denotation and constraints of placement and routing in the QCA domain are introduced. The P&R problem originally derived from [53] evolves from a grid enabling tile-based 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 $c(t^+) - c(t) \mod clk = 1$ accordingly. For QCA it was already stated that the clock number $clk = 4$.

From this definition we can outline the difficulty 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 true 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, *global synchronization constraint* states that every two signal paths leading to the same tile need to pass the same amount of tiles starting at their primary input. Since this constraint has to hold true 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 commonly accepted as $\mathcal{NP}$-complete [54].
After reaching a gate-level layout still a technology has to be mapped onto it. For this work the in subsection 2.2.2 proposed standard library is used for the mapping. Although the definitions in this work are quite generic because they are based on the book [53], defining the P&R problem for the domain of field-coupled nanotechnologies. This means that, for example, a change of clock to $clk = 3$ also allows a placement and routing for *Nanomagnet Logic* (NML). Even though the algorithm in this work is designed only for QCA, the ideas may also be derived for other FCN technologies.

2.4 Sequentiality

In the section 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. While combinational circuits can be

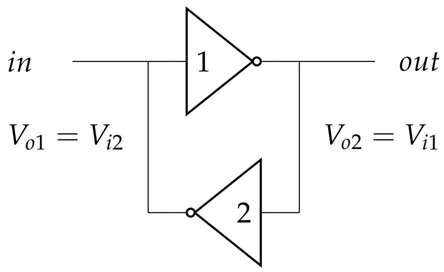


Figure 2.14: Eccles-Jordan-Flip-Flop

understood as combination of logic operations, sequential circuits additionally include the reuse of information computed by their logic, demanding 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 than in CMOS, storage elements and sequentiality have to be rethought. Hence, first the properties of storage elements are discussed in the CMOS domain and transferred to the QCA domain. It also has to be mentioned that the clocking in QCA only supports synchronous information flow, which means that in the first part also only synchronous CMOS logic is considered.

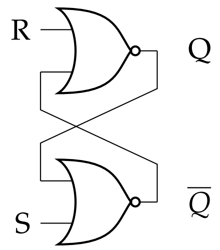
2.4.1 CMOS storage elements

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 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 sluggish.

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 transparency [18]. 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 a EJ-FF with NOR gates, the transparency gets reduced



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

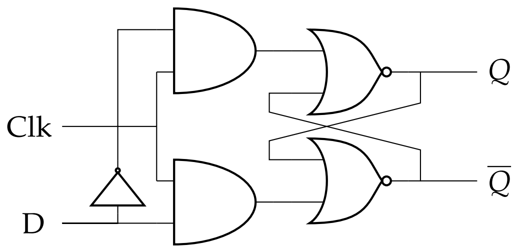
Figure 2.15: SR-Latch

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. Now a value is held when the clock $clk = 0$ and the D lock propagates the input value D when $clk = 1$. 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 D-FF can be constructed by connecting two D-latch stages behind each other. The master slave is activated at rising edges, while the slave 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 [18].

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



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

Figure 2.16: D-Latch

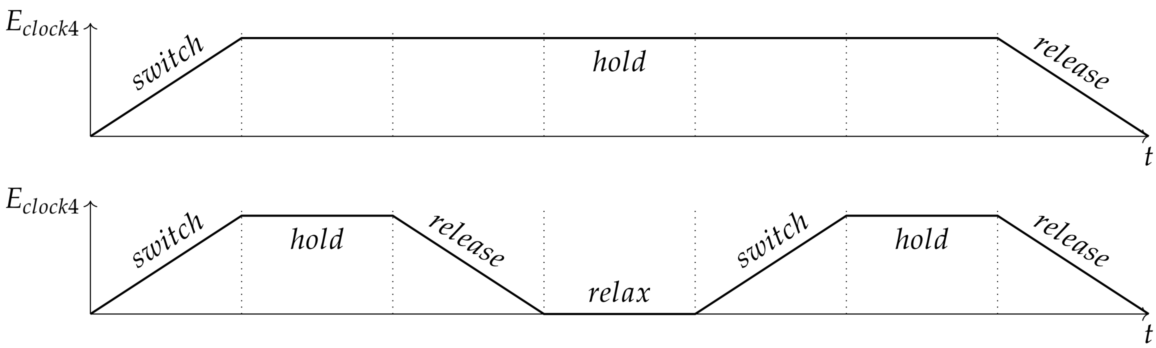


Figure 2.17: Clocking of a basic latch in QCA

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 from QCA to CMOS, as already observed in 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 [49]. 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. 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 the wire-FF misses is a set-reset option. The idea for this was also proposed in [53] by exchanging the wire with a majority gate but the same clocking. Now the D-FF basically has the same functionality but has 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.

3 State of the Art

In this chapter various approaches trying to solve the placement and routing problem for QCA are reviewed. 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 devised 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 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 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 [3]. 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 true or rather if they are consistent in the provided solution. If so, the abstraction

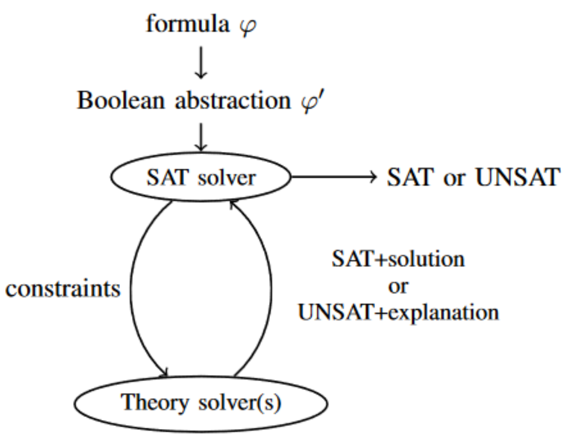


Figure 3.1: Lazy SMT-solving process [1]

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 [53]. The first algorithm "Exact Placement and Routing" 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 true. Therefore all possible sizes of layouts are encoded and passed to a SAT-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 [13, 51]. But due to the complexity of the algorithm utilizing satisfiability solvers, the algorithm times out for quite small circuits already, making it insufficient for the manufacturing of commercial QCA circuits.

The other exact P&R algorithm proposed in [53] 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 a 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 $|N| \geq 30$.
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 VLIS process, meaning they treat logic synthesis and physical design as their own problem and not as one-pass synthesis. Starting with logic synthesis, many works 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 reduce the number of fan-outs at the nodes, leading to a reduction of the place and rout 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 its marked as visited and if a already marked node is visited it is duplicated. This process is quite problematic, because not only the visited node is duplicated, but also all the nodes included in the sub tree rooted by it [47]. 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 don't have such a high overhead in dummy nodes but for that they can't eliminate all wire crossings, making it necessary to include nodes for crossings called *crossing edge insertions* [11]. Another preprocessing steps including the insertion of so called *buffer nodes* is aiming for the synchronization of 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. Also the insertion of buffers allows the generation of different partitions of a logic network [8]. Some approaches insert even a higher number of nodes in order to obtain a complete ternary logic network representation of a QCA circuit. 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

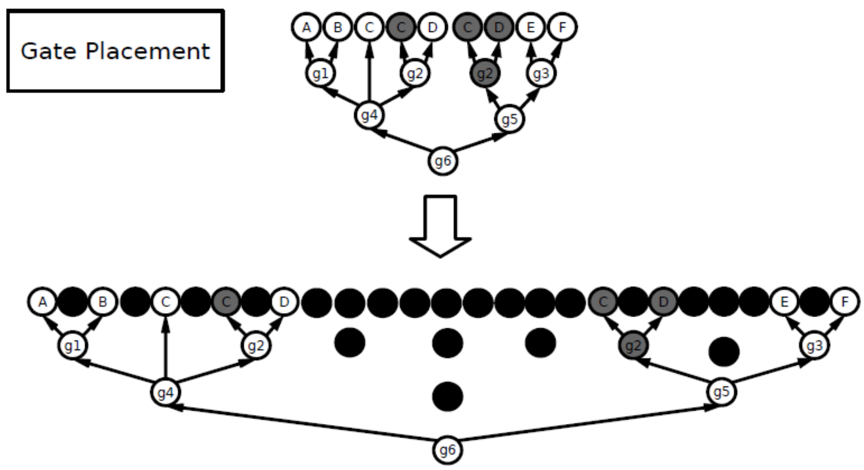


Figure 3.2: Gate placement with black circles showing wasted area [47]

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 [47].

Another big problem of these algorithms is the requirement of cell-based clocking itself, which has already been shown to be insufficient. Even though there also exist algorithms using tile-based clocking they are limited by the general drawbacks of preprocessing [51] leading to exploding logic networks and even use greedy placement and routing algorithms limiting the approach to small and simple reconvergent patterns [47].

All these reasons lead to the proposal of *ortho*, an algorithm implementing a scalable placement, routing and clocking without preprocessing steps proposed in [55]. Since this algorithms forms the base of this work, the algorithm is explained detailed in the following.

First of all, a proper representation of the logic network is needed. Therefore in some works already the idea of an orthogonal embedding, had been proposed [8]. Orthogonal embedding is the mapping if 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 [12].

Definition 3.1.1 (Orthogonal Graph Drawing)**.** An 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 from 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 OGD. The dots in the graph represent vertices and are

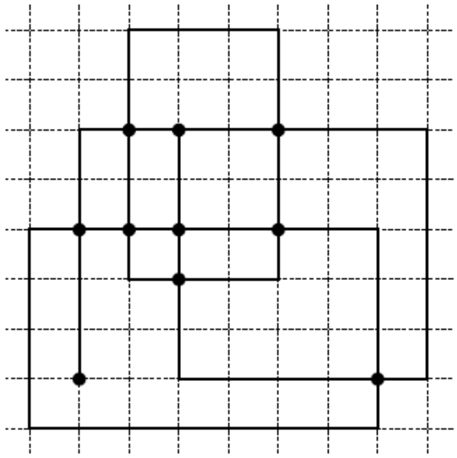
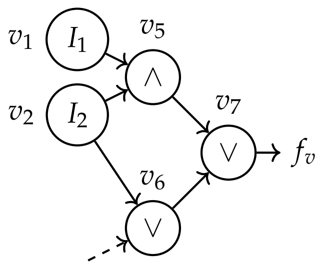
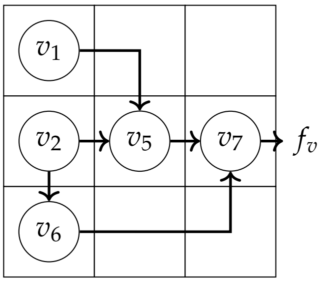


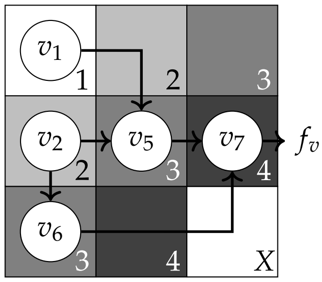
Figure 3.3: Example OGD drawing



(a) Partial representation of a logic network



(b) OGD representation of the logic network



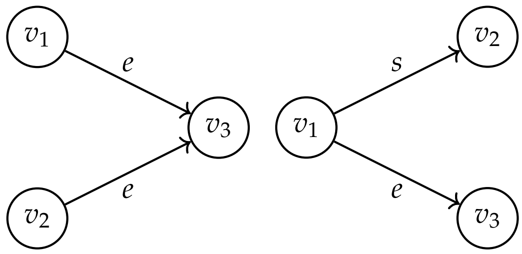
(c) No sufficient clocking for this OGD can be examined

Figure 3.4: Insufficient timing constraints of a OGD representation [53]

connected via straight line paths. Therefore the graph is drawn orthogonally.

Nonetheless an 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 shown from the example in figure 3.4. For the given OGD in subfigure 3.4(b) there has to be no clocking which can resolve the timing constraints. In subfigure 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



(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

the constraint needed for a scalable approach. The base used in [55] is formed by Therese Biedl [6], who proposes a 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 precious 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 own nodes for fanouts and inverters. 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 $d : \Delta \rightarrow \{east, south\}$ is assigned. The coloring can be understood as relative position arrangement. If an edge (v_i, v_j) is colored *east*, 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 true:

1. All **incoming** edge of a vertex has to be painted with the **same** color.
2. All **outgoing** edges of a vertex have to be painted with **opposite** 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. Figure !! shows such a coloring conflict. For the edge with a "X", no direction can be assigned for which the formulated constraints hold true. So 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 OGDs 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 [55] is depicted as algorithm 1 and described in own words, before its 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 for every tile in order to fulfill the timing constraints. As already mentioned N has to be converted into a 3-graph by substitution so a valid coloring can be assigned. Then an empty Layout L with a 2DD-Wave clocking scheme is created and the coloring is calculated for N. Also 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 under each other, but his leads to a conflict, when inputs are colored south, because the algorithm would then also wire their outgoing edges into y-direction and therefore over other primary inputs, which is forbidden in OGD and for QCA layouts. For this reason the primary inputs colored *south* need to be resolved by first rewiring them each on a new column before placing the nodes connected to their outgoing edges. 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. So 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 childs. 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 nodes childs 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 doesn't pass over another gate. If only one predecessor exists the wiring also goes only *south* or *east*. But when

two predecessors exist, in case of *east* the predecessor giving h_p is also only wired in x-Direction, while the 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 two segment 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.

Algorithm 1 Ortho algorithm

Input: Logic network N
Input: Clock number clk
Output: Gate level layout L

Convert N to a 3-graph by substitution
$L \leftarrow$ empty 2DDWave-clocked layout of size $w = 0 \times (h = 0)$
Generate direction assignment $d : \Delta \rightarrow \{east, south\}$ and subdivide signals if necessary
Compute topological ordering $v_1, ..., v_i \in N$
Extend L by one column and reserve it for primary inputs
for all vertex $v_1, ..., v_i \in N$ with at most two incoming signals σ_1, σ_2 **do**
    if vertex v is terminal/primary input **then**
        Extend L by one row
        Place v at position $(0, h-1)$
        if vertex v is colored $south$ **then**
            Extend L by one column
            Wire the primary input to position $(w-1, h-1)$
        end if
    else if $d(\sigma_1) = d(\sigma_2) = east$ **then**
        Extend L by one column
        $h_p \leftarrow$ max. vertical position of v's predecessors
        Place v at position $(w-1, h_p)$
    else if signals are labeled $south$ **then**
        Extend L by one row
        $w_p \leftarrow$ max. horizontal position of v's predecessors
        Place v at position $(w_p, h-1)$
    end if
    Extend L by one column and one row
    Wire the primary input to position $(w-1, h-1)$
end for
Draw orthogonal wire segments to connect v with its predecessor(s) accordingly
Connect the primary outputs to the respective borders **return** L

The example depicted in figure 3.6 shows the (p, r, c) of a 2:1 mux. Starting with a layout of size $1, 0$ 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

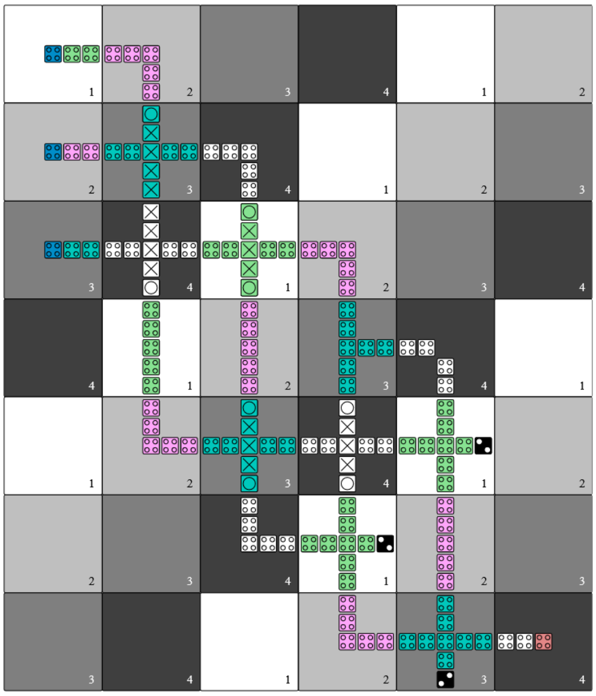


Figure 3.6: Placement and routing of a 2:1 mux network using the ortho algorithm

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 the input network is $(4,3)$. With v_4 a fanout 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 $L=(4,4)$. In the same fashion the parent node of the fanout v_5, representing an inverter is placed east of it. So the coordinates of the inverter are $(4,3)$ and $L=(5,4)$. Looking at the next node v_6, which is the first node with two children and which is labeled south now the x-coordinate is determined by the eastern predecessor, so $v_5=(4,3)$. 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 $(4,4)$ and $L=(5.5)$, The same way the AND-node $v_7=(3,5)$ $(L=(5,6))$ and the OR-node $v_8=(4,6)$ $(L=(5,7))$ are placed. Since all nodes are placed now the primary output has to be placed from v_8. For this the layout is increased by one column again $(L=(6,7))$ and the PO is placed to 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 the section about clocking 2.2.3, 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. Though, ortho provides an efficient tool for the placement and routing of purely combinational circuits. As shown in paper CITE, the 2DD-Wave scheme provides the most area efficient clocking when it comes to combinational circuits, beating both the USE and RES scheme.
With a deep understanding of the ortho algorithm with its constraints following some drawbacks and advantages, now other ideas with comparable approaches or based on the ortho algorithm can be discussed. *Ropper*, a placement and routing framework [14] proposes an algorithm, which is based on [51]. As already discussed in the part about preprocessing, this algorithm brings some disadvantages, because dummy nodes are inserted as part of logic synthesis, leading to an increased size of the logic network. In ortho this step is unnecessary. 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 ortho 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 and not like [56] suggests a minimum distance of two QCA cells. The use of these custom gates is necessary in this algorithm, because the design used doesn't rely on the same strict constraints as ortho. The Ropper framework even routes wires above gates (solved with custom tiles) and doesn't use border inputs and outputs making it challenging to input and read data from the circuit. Based on the argumentation used in this work, the Ropper framework violates to many design rules, which have been analyzed to be necessary for a sufficient placement and routing algorithm.
Another paper trying to implement majority gates for QCA is *migortho*, which is based on the ortho algorithm provided in fiction. The difference of the algorithms is the use of the underlying gate-library. While this work uses the same as ortho, migortho utilizes the QCA-ONE library. From the preliminaries, it is already known that again the use of rotated majority gates and therefore the use of double wires is allowed. 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. This fact has motivated this work to implement some different ideas to enable ortho to

be more area efficient, place "+"-majority gates and even implement a strategy for the automated placement and routing of sequential circuits.

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

Several attempts [29, 38, 44, 48] have been made to implement latches and FFs in order to obtain storage elements and enable sequentiallity for QCA circuits. The basic idea for these works is to translate the Boolean CMOS equations into majority representations and then implement this representation with QCA gates. Thus, the received circuit is on the one hand based on the 4-phase clocking of the QCA circuit and on the other hand demands an external 2-phase clocking signal adopted from the CMOS domain [29]. The authors of [38] even state that a latching can be accomplished using the QCA clocking, but that this restricts the circuit and therefore the external clocking signal has to be applied. According to the theory provided in the subsections about clocking 2.2.3 and storage elements 2.4 this doesn't hold true. The rethinking of these elements in the QCA domain even suggests the use of the already provided 4-phase clocking to accomplish sequential functionalities. This observation seems to be clear, considering that the clock used for synchronization in the storage elements should also drive signal propagation. Since in QCA the 4-phase clock is responsible for this, the use of an external clocking signal is questionable and invalid for this work. Although this state of the art already provided the proposal of far developed sequential circuits, e.g. dual edge triggered D-FFs [44] and reversible latches [48] it also is found that these works all use cell based clocking, which was already found to be insufficient in 2.2.3. The evolution to sequential tile based placement and routing was proposed through the USE clocking scheme [9], supporting back-looping. The paper shows as ideas the layout of a SR-latch and a full adder. Unfortunately the SR-latch is also implemented in the way as described above and the implementation of combinational logic was found to be worse for most benchmark problems than for 2DD-Wave [53]. The works [34] and [5] propose implementations of different latches and even an algorithm of implementing sequential elements based on the USE scheme. Still these approaches all share the shortcoming of translating sequential logic directly from the CMOS domain.

As already stated in this work wire segments should be used as storage elements as proposed in [53]. Unfortunately this work doesn't provide any circuit or a placement and routing algorithm using these elements. In this work the basic ideas of wire delays is used and adjusted according to the placement and routing of QCA circuits.

3.2.2 QCA storage cells (QCA RAM)

Another section of papers focuses on the implementation RAM cells in QCA technology. Even though this paper focuses on a placement and routing algorithm for sequential logic, also the implementation of QCA storage could be adapted. The state of the art can be devised into two approaches of implementing RAM in QCA. In the papers [10, 56, 45] different RAM implementations are presented, translating again CMOS technology into QCA and therefore dealing with the same shortcomings resulting from an external clock signal. Also the circuits are again based on cell-based clocking, being insufficient for this work. The second part of papers [2, 31] proposes the use of MUX structures in order to implement RAM cells. Considering a 2:1 MUX with a RAM cell holding one bit of information and a bitline (BL) with also one bit as inputs, the 2:1 MUX can decide if the information on the BL is passed into the RAM cell or if the information in the RAM cell is held. The decision is taken by the information on the wordline (WL), which is the third input to the 2:1 MUX. Unfortunately the implementations shown in [2] and [31] still use an external clock signal and are also clocked cell-based. Nevertheless, the ideas of wire delays is combined with the idea of using MUX to build a RAM cell, which can be built using ortho and the corresponding sequential distribution network, which is later proposed in this work.

4 Methodology

This chapter proposes three different signal distribution networks, which overcome the restrictions of the placement and routing algorithm ortho, reviewed as state of the art in chapter 3. Therefore, an ordering, a majority gate, and a sequential distribution network are introduced. Since ortho has been shown to be restricted, but yet has a very powerful placement and routing procedure, the base of the algorithm should be maintained while implementing new functionalities as irregularities. The task of the signal distribution networks is to redistribute signals on the layout in a way that the irregular parts fit with the regular placement and routing, while all design constraints are retained. In this way, the algorithm can still work in a similar fashion as ortho. However, also the underlying logic network and the placement and routing itself have to be modified in order to add all the desired functionalities, adding complexity to the preprocessing and the algorithm itself. Because ortho is restricted to the use of only 2DD-Wave clocking, signal distribution networks need the power to change the clocking in the layout to implement the functionalities they provide. Hence, signal distribution networks must be implemented with care and synchronization constraints must be considered very closely. The ordering distribution network is discussed below 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

Looking again at the resulting layout of a 2:1 mux or ortho in 3.6, it can be seen that in the first few rows, where the primary inputs are placed, no other gates are placed, because the space has to be reserved for rewiring in order to solve conflicts. The idea of the ordering distribution network is to allow gates to also be placed in this area to save space and to place inputs in a way that wire crossings may be minimized. To do so the ordering network has to resolve the conflicts in some other way. Recalling the pseudocode from the ortho algorithm, an input has a conflict when it is colored south because it is not allowed to wire over the other inputs laying in the same column. This means the area overhead in the input region is highly dependent on the coloring assigned to the outgoing edges of the inputs. The algorithm used in the pseudocode line 3 is

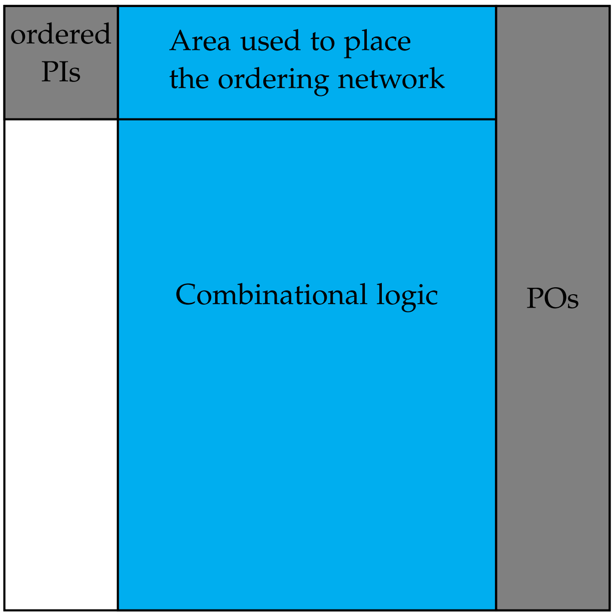


Figure 4.1: Scheme of area usage in the Ordering Distribution Network

implemented in a way that finds just *some* valid but not an *optimal* coloring for the given logic network. Unfortunately due to the algorithms nature it often assigns the color south to exactly these edges resulting in the said area overhead. Therefore, the first step is to improve the coloring of the logic network and prevent excess wiring. Secondly, a new rule for edges colored south inside the conflicting area can be introduced, making the rewiring redundant. The third idea which can be implemented in the network is an ordering of the inputs in order to allow those who are connected with the same gates to be placed near each other reducing wire expenses and crossings.

Algorithm 2 Ortho changes with ordering distribution network

⋮

Convert N to a 3-graph by substitution and balance inverters at fan-out nodes
Order primary input nodes
⋮
Generate **conditional** direction assignment $d : \Delta \rightarrow \{east, south\}$ and subdivide signals if necessary
Compute topological ordering $v_1, ..., v_i \in N$
Extend L by one column and reserve it for primary inputs
for all vertex $v_1, ..., v_i \in N$ with at most two incoming signals σ_1, σ_2 **do**
  if vertex v is terminal/primary input **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

To discuss the idea, first the parts of the logic network, which belong to the ordering distribution network have to be determined. This can be deduced by looking at the layout, which shows the wasted area for mainly the primary inputs and nodes connected with them. Thus, all primary inputs and the gates which they are wired to, skipping over inverters, are viewed as part of the ordering distribution network. Skipping means that if the outgoing edge of an primary input is an inverter, the gate hanging on the outgoing edge of the inverter is considered. Starting with the different gates inputs can be connected to, the coloring they can have should be discussed. The direction assignment of one-input nodes, including inverters and fan-out nodes, can be chosen arbitrarily because the primary input to which they are connected has always only one outgoing edge, resulting in no dependencies. In this case, always the non-conflicting *east* assignment can be chosen. When looking at two-input logic gates like AND and OR gates it has to be seen that the coloring can only be chosen arbitrarily if both input nodes are primary inputs, allowing again the non-conflicting assignment of *east*. In every other case, the direction assignment has to consider the coloring of the other incoming edge of the gate. In order find out the dependencies, the ordering distribution network places first every primary input connected to a fan-out node with its respective fan-out node itself into the layout. This has several advantages. First of all, the fan-out nodes give the constraints for the conditional coloring, which is introduced in the ordering distribution network. Also fan-out nodes produce new paths and therefore excess wiring, which means that their dependencies should be resolved as fast as possible by placing and routing them to their outgoing gates as fast as possible. Following the coloring rules, two outgoing edges of a node 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 the other incoming edge of the gate connected to the colored edge *south*, also to be colored *south*, and the second incoming edge being connected to a primary input, we can see that a conditional coloring alone is not powerful enough to resolve all conflicts. For this case a new placement rule for the *south* coloring is introduced in order to preserve the direction assignment rules but still resolve the conflict between primary inputs. The original 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 too be to restrictive, allowing the following placement rule for *south*. 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 column while the gate is still in position $(w_p, h-1)$. Following this rule the gate is now placed in the same column as its predecessor with the higher y-coordinate. If we apply this to a two-input gate in the ordering distribution network with a primary input and a fan-out node as predecessors, the primary input is always the root node due to the ordering and new coloring. Thus, the new rule allows the two-input gate connected to the primary input colored *south* and the fan-out node to be placed in the same column as the primary input, 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 this special case, but also for the general *south* placement in the algorithm with one exception. 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. The resulting pseudo-code snippets replacing the used code are shown in algorithm 2. Also it has to be considered that the conditional

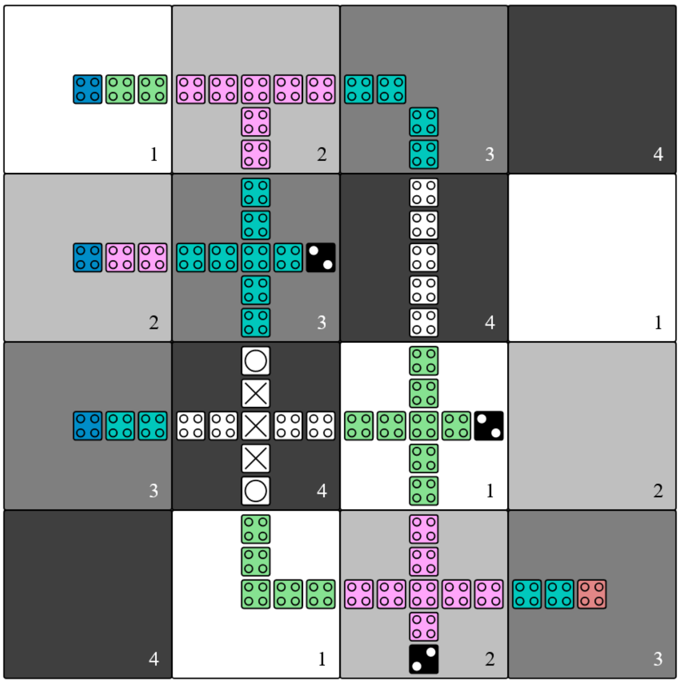


Figure 4.2: Placement and routing of a 2:1 mux network using the ortho algorithm with the ordering distribution network

coloring in the distribution network still needs to include helping nodes e.g. when three fan-out nodes are connected to each other. Also before the coloring, first the input nodes need to be ordered according to the ideas presented. Thus, primary input nodes connected to fan-out nodes are placed first and then the primary input nodes, which are connected to the outgoing edges of the fan-out nodes are placed. This is done to reduce the distance between coherent gates and therefore also the number of wire crossings. Afterwards primary inputs directly connected to a gate which has its other incoming edge connected to a second primary input are placed. Finally all input nodes, which are not connected to the rest of the ordering distribution network are placed arbitrarily and the logic network is topologically ordered according to the new order of the primary inputs. Some other issues are related to inverter nodes. As already mentioned they are skipped in the view of the ordering distribution network, but still need to be considered for the placement and routing. Assuming an inverter node which is assigned *south* e.g. after a fan-out node and it should be placed in the same row as an primary input, a conflict arises because the input always has to wire east first. Thus, all inverters colored *south* need to be placed to minimum the row of the most southern primary input plus one. In order to prevent to much overhead produced by inverters a balancing network is introduced, which aims to reduce the number of inverters in the logic network. Based on the substitution of the logic network into N with inverter and fan-out nodes in some cases a fan-out node has two inverters connected to its outgoing edges. Then these inverters are substituted by one single inverter as incoming node to the fan-out, resulting in an overall lower number of inverter nodes. Figure 4.2 shows the placement and routing of the ortho algorithm after implementing the proposed ordering distribution network. The ordering of the inputs puts first the fan-out node and then the two connected primary inputs. In this case the ordering distribution network also considered the inverter at the outgoing edge to be colored east in order to produce less overhead, because if the inverter would be colored *south* it would have to be placed underneath the primary inputs. Looking at the AND gate connecting the fan-out with the second primary input, we can see that the new rule for nodes placed south is used. This also applies for the AND gate connecting the third primary output to the inverter. The last OR gate is placed after the normal rules of the ortho algorithm. In the comparison to the layout in figure 3.6 can be quickly seen that the resulting layout saves up place and even wire crossings. The exact results are presented and analyzed in the next chapter.

4.2 Majority Gate Distribution Network

In this section the placement and routing of majority gates, using the ortho algorithm is discussed and a distribution network is proposed. The placement and routing of majority gates plays a major role in QCA since the majority function can be implemented using only one gate in contrast to a CMOS implementation using multiple gates. But this theoretical advantage can only be exploited, if an efficient placement and routing exists. To this use a distribution network for ortho is introduced, allowing a comparison of design metrics after placing and routing a logic network using the majority gate distribution network with a logic network placed and routed using the QCA implementation without majority gates.

4.2.1 The proposed signal distribution Network

The 2DDWave clocked ortho algorithm only supports the placement and routing of 2-input logic gates and therefore the direction assignment contains only two directions *east* and *south*. Since the goal of the distribution network is to introduce "+"-majority gates into the layout a RES-like clocking needs to be utilized, introducing tiles with three incoming tiles and one outgoing tile. In the RES scheme in figure 2.13(c) such a tile is on position $(1, 1)$ and would be suited to place a "+" majority gate on it and allowing it to be connected with three incoming signals. However, simply changing

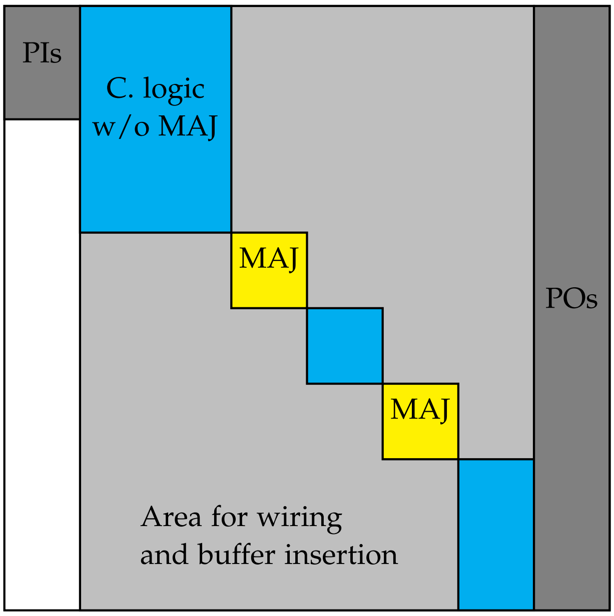


Figure 4.3: Scheme of the P&R using the Majority Gate Distribution Network

the clocking scheme of ortho to RES would be really inefficient and could not be easily implemented. On the one hand, if the clocking would be completely changed to RES, the algorithm could not utilize every row and column of the clocking since the RES scheme also supports signals to flow into western or northern direction. In RES only the first and third row such as the second and fourth column support eastern and southern signal propagation, so only these part of the clocking would be utilized for the placement of two input gates, which should lead to about a doubling in area usage considering only two-input logic gates. On the other hand, for the placement of three input majority gates a new direction assignment would have to be introduced to the logic network, since one signal would need a direction assignment *west*, changing the theory and complexity underlying the ortho algorithm. Another idea utilizing the RES scheme would be to only support it in some evenly distributed regions, supporting majority gates just at some permanently assigned places. For this the layout could be devised into 4x4 sub-regions and e.g. every fifth sub-region would be RES clocked and the rest would be occupied with the 2DDWave scheme. On the one hand this realization should not produce that much area overhead since only some regions are inaccessible for two input logic gates, but the permanent clocking assignment only allows the placement of majority gates gets forced on some permanent spots, leading again to large area overhead if a majority gate should be placed far away from such a sub-region. Another consideration would be if a network with only majority gates should be placed in this implementation all the 2DDWave clocked area would be wasted. Another aspect the ortho algorithm makes use of is the trivial global synchronization constraints within a uniformly 2DDWave clocked layout. By introducing irregular clocking e.g. RES sub-regions, signals can pass a different amount of tiles in order to reach the same tile, therefore violating the global synchronization constraint. Figure !! a RES sub-region embedded in a 2DDWave clocked layout. Drawing the paths from three synchronous starting points to the three input tile, it can be seen, that the paths have different lengths, violating the design rules.

To avoid these complications, the proposed distribution network utilizes a custom clocking only in areas where majority gates are placed and find a solution for the global signal synchronization constraint. Therefore the placement and routing of solely two input gates should not produce any excess area. Figure 4.4 shows the proposed majority gate signal distribution network, which is implemented into the ortho algorithm. The red marked cells indicate the three inputs for the majority gate distribution network. The cells in the middle of the tile are marked, because they can be connected from above north or the west, which would result once in a normal wire and once in a bent wire. The output in blue enables the algorithm to wire it in east or in south direction, hence allowing the algorithm to work without limitations. It is also important to mention that the input tiles as well as the output tiles have the same clocking number as in a regular 2DDWave scheme, allowing straight forward connections. Although no area overhead is produced for the already existing placement and routing of two-input gates, it can already be seen that the distribution network itself produces excess area due to its complex wiring, resulting again from the synchronization conditions that had to be considered designing it. The implementation of an AIG representation of the majority function implemented with ortho is depicted next to the distribution network to visualize that the placement and routing of this single majority gate barely safes area, although it has to be highlighted that no wire crossings are used. Under the assumption of really high costs for wire crossings the found majority gate distribution network is considered to be more ideal, even though a meaningful cost comparison of these two implementations can only be done under a cost function representing wire crossings in cost of normal gates. In the following the design constraints used to design the signal distribution network are discussed. Firstly the distribution network should not contain any wire-crossings, since they are considered to be very costly. Introducing a cost-metric for wire-crossings may result in a more efficient implementation, but for this work wire-crossings should be excluded as design rule for the network. Secondly, the distribution network needs to meet the global synchronization constraint. Considering a 2DDWave clocked layout, every diagonal is synchronous and every signal wired on the same diagonal passes the same amount of tiles following the ortho placement

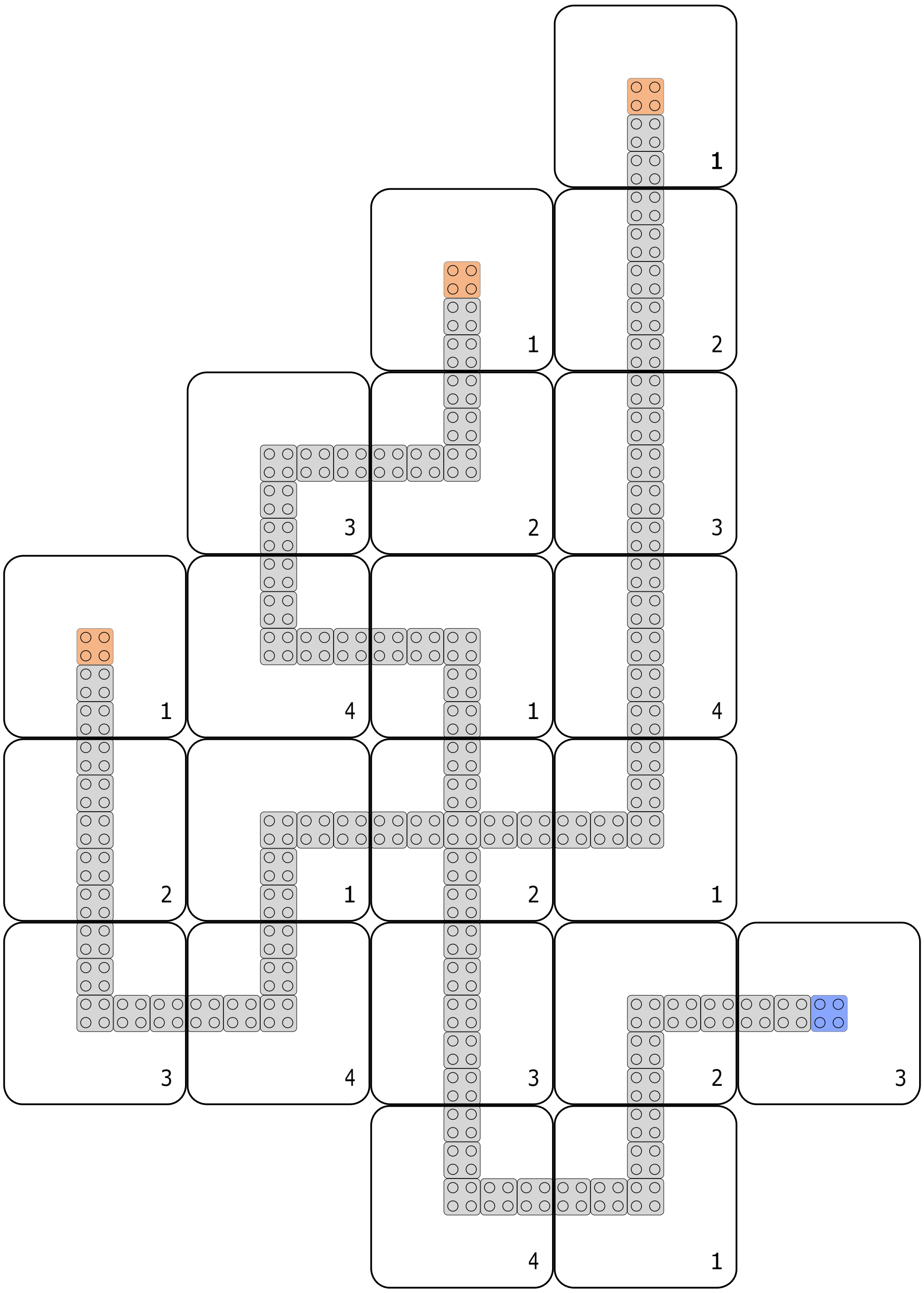


Figure 4.4: Proposed majority gate distribution network

and routing. If we look at the incoming tiles of a three input tile, it can be seen that only two of the incoming tiles are on the same diagonal and the third one is shifted by half a clock cycle. Hence the third incoming signal is delayed by half a clock cycle violating the global synchronization constraint. Also the signals need to pass a multiple of whole clock cycles in the signal distribution network in order to support the further use of 2DDWave and the local synchronization constraint. Therefore, first the initially synchronous signals are also delayed by half a clocking signal, satisfying the global synchronization constraint at the tile, where the majority gate is placed. Afterwards the output signal of the majority gate is again delayed by half a clock cycle so it can be connected to the regular 2DDWave clocking scheme used in the remaining layout. Adding up the delays resulting from the distribution networks leads to a total delay of one whole clock cycle of the signal propagating through the majority gate distribution network compared to all other signals in the logic network. Because the delay affects the gloal synchronization constraint, it has to be considered for each gate connected to the parents of a majority gate distribution network. In the following first the basic placement and routing of the majority gate distribution network and then the solution for meeting the global synchronization constraint in further placement and routing is discussed.

4.2.2 Placement and routing

The placement and routing of the proposed signal distribution network is again bound to some constraints. First of all the coloring of majority gates has to be reviewed, since the logic network now includes three input nodes. However, the coloring algorithm can include helping nodes to resolve coloring conflicts of edges and therefore dividing every edge with a helping node shows, that a trivial coloring can be found also including three input nodes into the logic network. Another aspect regarding coloring is the need for a new direction in order to connect a third signal to the majority gate, but since the only occasion such a wiring happens is inside the fixed distribution network, which again can be placed and routed in the usual south-eastern manner, no additional directions need to be included. Further, the irregular clocking inside the signal distribution network has to be reviewed. These irregularities don't allow the algorithm to wire connections over the network, demanding a special treatment for the placement of the majority gates. From the algorithms perspective therefore a majority gate cannot be placed just south or east of another gate because these gates could need a wiring through the majority gate distribution network. Instead the algorithm is forced to assign the majority gate distribution network always south **and** east direction to prevent routing conflicts. This can be done by dividing the incoming edges with auxiliary nodes, also allowing a valid coloring. This also allows the coloring to be included into

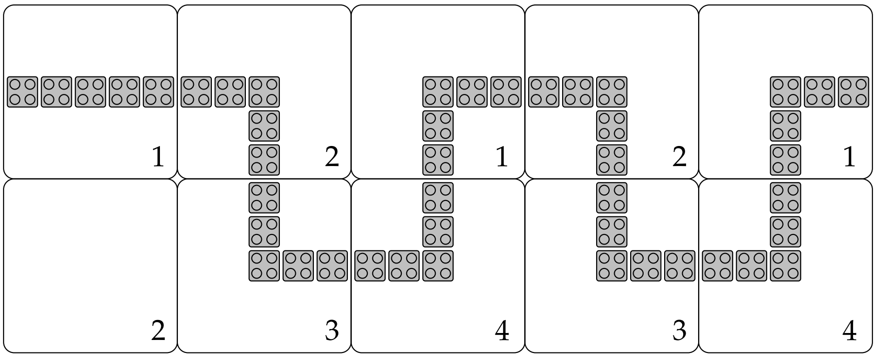


Figure 4.5: Buffer in *east* direction with resolve column and respective clocking

the input ordering distribution network, so that all inputs connected to a majority gate need to be colored *east*. The major drawback of this is, that again the area is not used optimally and the layout is extended in two directions compromising the beneficial use of the "+"-majority gate.

4.2.3 Signal synchronization and buffer insertion

The placement and routing using the proposed distribution network results in a delay of one clock cycles of signals passing through a majority gates. Since the tile-based clocking doesn't support a speedup of a signal, every other signal which comes into contact with a delayed signal also has to be delayed. 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 primary output. 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 again is not connected to a majority gate. In the latter case all incoming edges of a node would be delayed, resulting again in a synchronous behavior. Hence the delay on one node can be maximum one clock cycle. After computing the delayed edges, they are realized by inserting wire buffers.

Figure 4.5 depicts a buffer in the east direction, which can also be used in the south direction by just rotating it 90 degrees. The snake-shaped structure delays a signal by exactly one clock cycle and is also used in custom placement and routing resulting from the QCA ONE library [37]. As in the majority gate distribution network, 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, allowing one to track them and 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 majority gates distribution networks and the corresponding majority buffers. Figure 4.6 shows the placement of a majority gate inside the input distribution network and two and gates that have to be delayed to be connected to the delayed signal coming out of the majority gate distribution network. The insertion of the first buffer blocks the eastern direction of the second input. For this case, a resolve column is introduced where the signal can be assigned to a new row and be wired without conflict. From this layout, it can already be seen that the implementation of the majority gate distribution network brings several complications with it, all resulting in area overhead, which stands in contrast to the area which should be saved by introducing majority gates in the first place.

Algorithm 3 Ortho changes with majority gate distribution network

⋮

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 σ
Order primary input nodes
Create vectors with from the majority buffers blocked columns bl_c and rows bl_r
⋮
for all 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 fanin 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 majority gate distribution network and place the distribution network at $(w-5, h-7)$
        Connect incoming signals west to the inputs of the distribution network
    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

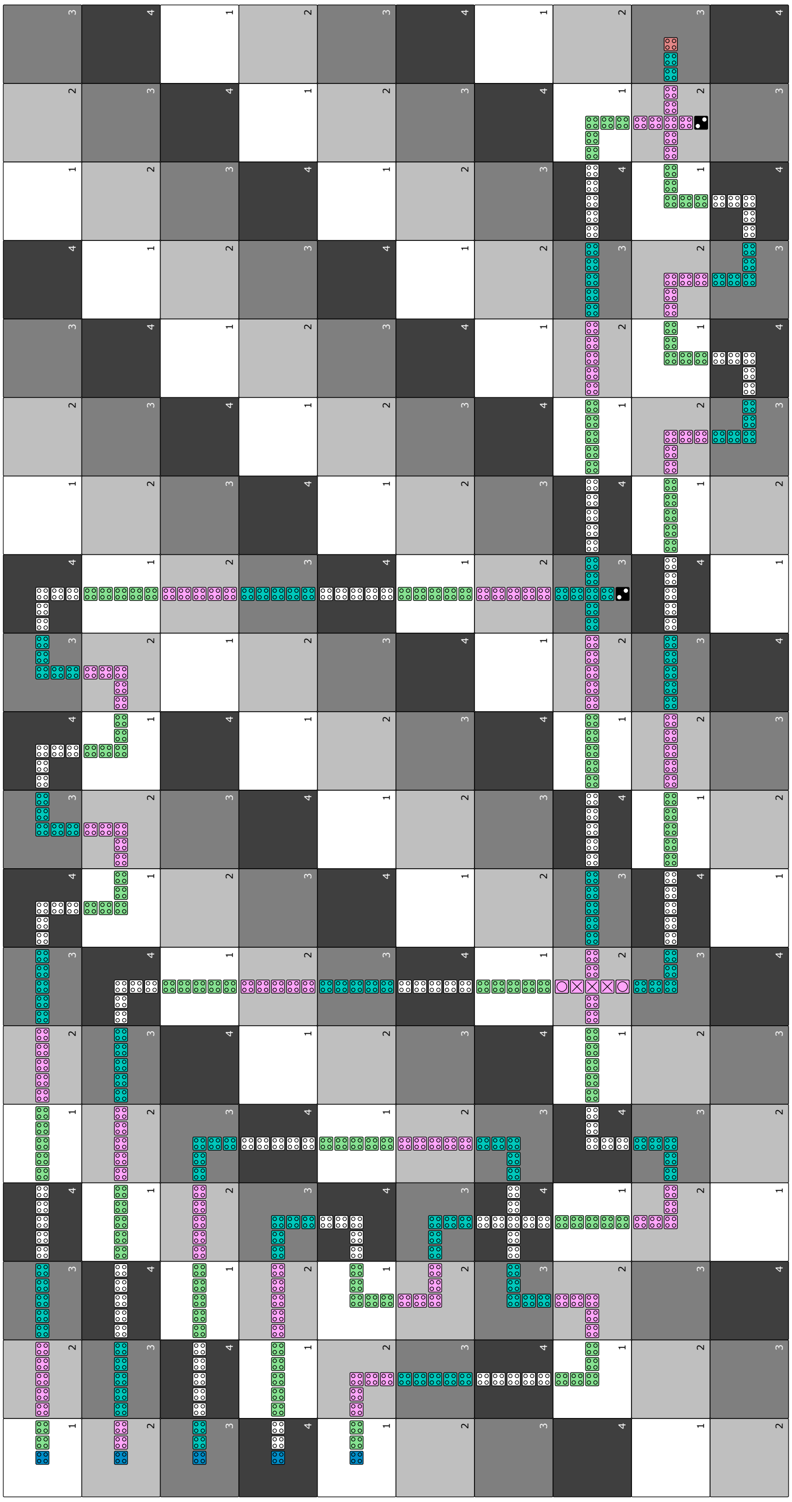


Figure 4.6: Placement and routing of a majority distribution network in conjunction with two primary inputs

4.3 Sequential Distribution Network

4.3.1 Placement and Routing

In this section, a distribution network is provided, which enables ortho to automatically design sequential circuits. To the authors' knowledge, there has been no solution to place and route sequential circuits in QCA yet. The only algorithms dealing with sequentiality in QCA, presented in Chapter 3, only translate CMOS structures directly into QCA and therefore require an external clock signal, which was stated to be unnatural, because QCA has its own clocking paradigm. For the placement and routing used here, the ideas of gates, able to delay signals from [53], was used and further developed for the automated placement and routing.
Before discussing the distribution network, first the idea of an FF wire should be discussed in the domain of placement and routing and not only as a single element. From the FF wire can be examined that the proposed FF implementation requires more complex clock generators for every FF, and it is not sure if this is possible to implement. Also if we look back at the analogy of CMOS now the sequential circuit includes a combinational logic block and the storage element formed by a wire FF. But there is still a big difference. While in CMOS the information can just be arbitrarily wired back from the storage to the inputs of the combinational logic, in QCA the wiring back implies the placement of wire segments of which each is delaying the information by one clocking zone already, being a partial FF. If a signal is wired through four adjacent wire gates, a basic FF is already formed, since the information is delayed by four clock zones equaling a clock cycle. When looking back to the functionality of a storage element, it can be found that this delay is exactly the purpose of a clocking element and the reason why the clocking for the wire FF is customized. The idea proposed in this work is now to use the delay, which occurs naturally due to sequential wiring to mimic storage elements and therefore wire segments can be summarized to FFs without the need for customized clocking.

Considering the placement and routing of sequential circuits, not only a distribution network has to be designed but also the logic network has to be expanded regarding to storage elements. They are represented in the logic network by registers with its corresponding input, determining the value, which has to be stored, and its output, which gives the register value to the combinational logic again after delaying it to the next *circuit clocking cycle*. A circuit clocking cycle refers to one cycle of Bennet clocking that has propagated through the circuit completely. The registers are implemented into the logic network as follows. Register inputs (RIs) are treated similar to primary outputs, therefore they are dangling edges, which point to no node but additionally have a register output assigned. Register outputs (ROs) are treated similarly to primary

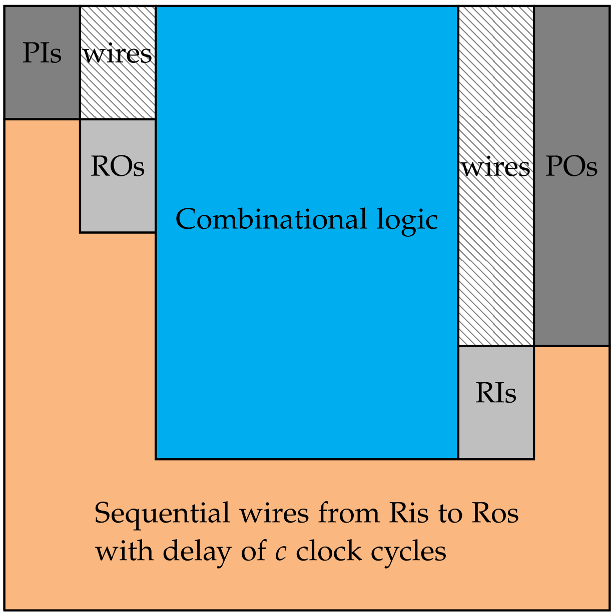


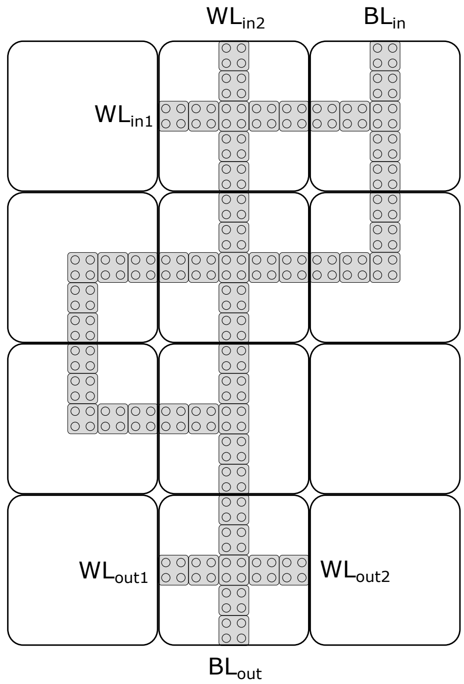
Figure 4.7: Scheme of a sequential circuit layout after placement and routing

inputs, being terminal vertices, but always feeding in the data which were given to the corresponding register input in the last circuit clocking cycle. Therefore, the logic network extends to $N = (\Lambda, I, RO, \Sigma, O, RI)$. Here it has to be mentioned that for the input distribution network due to their similarity ROs can be treated just like PIs, enabling the combination with the sequential distribution network. 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 4.7. When the first part of ortho is performed, first the combinational logic part is placed and routed, treating ROs just like PIs and RIs just like POs. 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 register input has exactly one register output assigned, first of all, the register inputs are rewired and sorted in the same order as the register outputs. 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 register input and output. 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. One big issue is also that the clocking cannot be chosen independently for each back-loop because the

loops cross each other. The solution found can be seen in an example depicted in figure !!. From this example another issue regarding timing can be derived. Considering a primary Input in a completely combinational circuit being placed in the fifth row of the layout. In this case the PI is set in a different *time zone*, 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 primary input by one clocking cycle to achieve again global synchronization. When an RO is placed in a different time zone, 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 clocking cycle but by multiple clocking 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 sequential signal distribution network always grows with the size of the combinational logic.
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 sequential distribution network can be decreased as well.

4.3.2 RAM cell

As mentioned in 3.2.2, a RAM cell can be realized using wire delays to safe the data and a MUX, which can input new data into the RAM cell. With the possibility of placing and routing sequential circuits a RAM cell now is a MUX with two PIs and one RO. The PIs are the BL and WL, while the RO is the information held in the RAM cell. The output of the MUX is both the PO and the RI. Figure 4.8 shows a RAM cell, with its respective wordline(s), bitline(s) and latch. The core of the cell is a latch, which is simply propagating the data in a circle and therefore producing a stable output once in every clock cycle. The input mechanism works via two majority gates. The majority gates with the wordlines decides if BL or $\bar{BL}$ should be propagated to the majority gate connected to the loop. When $WL_1 = \bar{WL_2}$ the majority gate always outputs the third input BL. But for $WL_1 = WL_2 = \bar{BL}$ the first majority gate outputs $\bar{BL}$. If the first majority gate has as output BL, the second majority gate has BL twice as input and overrides the data in the RAM cell. If the first majority gate has as output $\bar{BL}$, the second majority gate has two distinct inputs and the data which is held in the cell. In this case the stored bit always decides the output and therefore it is latched meaning that the data is stored. For the read operation only one majority gate is needed and in order to read the RAM cell the output-wordlines have to be inverted. Otherwise if both output-worlines are set to "0", the output isn't read. The corresponding truth tables for the RAM cell is shown in 4.8(b).



(a) QCA implementation

	Input1	Input2	Input3	Output
Maj1	WL_1	$WL_2 = \overline{WL_1}$	BL	BL
	$WL_1 = \overline{BL}$	$WL_2 = \overline{BL}$	BL	$\overline{BL}$
Maj2	X	BL	BL	BL
	X	$\overline{BL}$	BL	X

(b) Truth table

Figure 4.8: QCA RAM cell using wire delay

5 Experimental Evaluation

5.1 Benchmarks

For combinational and sequential

5.2 Results

	Benchmark			Ortho					Ordering NW				
	Name	I/O	$\lvert G \rvert$	w	h	$\lvert WC \rvert$	$\lvert W \rvert$	t in s	w	h	$\lvert WC \rvert$	$\lvert W \rvert$	t in s
trindade16	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	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	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					Ordering NW				
	Name	I/O	|G|	w	h	|WC|	|W|	t in s	w	h	|WC|	|W|	t in s
EPFL	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

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

Benchmark			Ortho						MAJ DNW					
Name	I/O	|M|	|G|	w	h	|WC|	|W|	t in s	|N|	w	h	|WC|	|W|	t in s
r1	3/2	5	84	29	64	92	709	<1	26	107	73	70	1440	<1
r2	4/3	7	90	30	72	92	731	<1	28	176	87	78	2377	<1
r3	6/10	30	432	138	318	1205	11838	<1	132	904	385	831	34158	1,44
r4	10/27	93	1321	390	980	8479	100222	2,42	350	3722	1000	4813	355351	12,91
r5	10/50	190	2668	765	1975	31667	371272	14,68	712	8112	1978	16555	1431295	96,20
r6	10/25	85	1204	361	890	6803	79105	2,01	310	2744	902	3915	235040	9,71
r7	23/70	267	3799	1106	2811	63270	744287	22,72	971	11421	2801	32302	2791464	111,13
r8	16/85	324	4638	1321	3436	86195	1050366	29,24	1189	13714	3349	41489	3943616	194,62
r9	16/85	61	864	253	643	3907	40424	1,12	245	1815	665	2280	120456	6,82
r10	8/21	83	1149	333	855	6861	74636	1,89	306	2706	854	3623	235136	9,74
r11	12/46	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, |M| number of majority gates, |G| number of logic network nodes (gates + fan-outs), w × h aspect ratio given in tiles, |WC| number wire crossings, |W| number of wires, w × h aspect ratio given in tiles, w × h aspect ratio given in tiles,t in s runtime in seconds, OOM maximum RAM reached, —no data available

	Name	I/O	|R|	|G|	w	h	|WC|	|W|	t in s
itc99-poli	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

List of Figures

List of Tables

Bibliography

[1] E. Ábrahám and G. Kremer. "Smt solving for arithmetic theories: Theory and tool support." In: *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE. 2017, pp. 1–8.

[2] F. Ahmad. "An optimal design of QCA based 2n: 1/1: 2n multiplexer/demultiplexer and its efficient digital logic realization." In: *Microprocessors and Microsystems* 56 (2018), pp. 64–75.

[3] 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, pp. 268–283.

[4] Z. Beiki and A. Shahidinejad. "An Introduction to Quantum Cellular Automata Technology and Its Defects." In: *Reviews in Theoretical Science* 2 (Dec. 2014), pp. 334–342. DOI: 10.1166/rits.2014.1028.

[5] 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), p. 107668.

[6] T. Biedl and G. Kant. "A better heuristic for orthogonal graph drawings." In: *Computational Geometry* 9.3 (1998), pp. 159–180.

[7] R. E. Bryant. "Graph-based algorithms for boolean function manipulation." In: *Computers, IEEE Transactions on* 100.8 (1986), pp. 677–691.

[8] 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, pp. 243–248.

[9] 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), pp. 513–517.

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

[11] 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), pp. 497–505.

[12] M. Eiglsperger, S. P. Fekete, and G. W. Klau. "Orthogonal graph drawing." In: *Drawing Graphs*. Springer, 2001, pp. 121–171.

[13] 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: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2018, pp. 1–5.

[14] R. E. Formigoni, R. S. Ferreira, and J. A. M. Nacif. "Ropper: A placement and routing framework for field-coupled nanotechnologies." In: *2019 32nd Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE. 2019, pp. 1–6.

[15] A. Gin, P. D. Tougaw, and S. Williams. "An alternative geometry for quantum-dot cellular automata." In: *Journal of Applied Physics* 85.12 (1999), pp. 8281–8286.

[16] 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), pp. 83–96.

[17] A. Grabowski. "Mechanizing complemented lattices within Mizar type system." In: *Journal of Automated Reasoning* 55.3 (2015), pp. 211–221.

[18] 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. ISBN: 9781613530023.

[19] E. V. Huntington. "Boolean Algebra. A Correction." In: *Transactions of the American Mathematical Society* 35 (1933), p. 557.

[20] 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), pp. 179–180. DOI: `10.1073/pnas.18.2.179`. eprint: `https://www.pnas.org/doi/pdf/10.1073/pnas.18.2.179`.

[21] P. T. Johnstone. "Conditions related to De Morgan's law." In: *Applications of sheaves* (1979), pp. 479–491.

[22] R. W. Keyes and R. Landauer. "Minimal energy dissipation in logic." In: *IBM Journal of Research and Development* 14.2 (1970), pp. 152–157.

[23] 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), pp. 1–15. DOI: `10.1007/s10470-018-1270-x`.

[24] R. Landauer. "Irreversibility and heat generation in the computing process." In: *IBM journal of research and development* 5.3 (1961), pp. 183–191.

[25] 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), p. 4240.

[26] C. S. Lent and P. D. Tougaw. "A device architecture for computing with quantum dots." In: *Proceedings of the IEEE* 85.4 (1997), pp. 541–557.

[27] 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. PhysComp'94*. IEEE. 1994, pp. 5–13.

[28] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein. "Quantum cellular automata." In: *Nanotechnology* 4.1 (1993), p. 49.

[29] L. A. Lim, A. Ghazali, S. C. T. Yan, and C. C. Fat. "Sequential circuit design using Quantum-dot Cellular Automata (QCA)." In: *2012 IEEE International Conference on Circuits and Systems (ICCAS)*. IEEE. 2012, pp. 162–167.

[30] M. Mahdavi, M. A. Amiri, S. Mirzakuchaki, and M. N. Moghaddasi. "Single Electron Fault in QCA Inverter Gate." In: *2009 Fifth International Conference on MEMS NANO, and Smart Systems*. 2009, pp. 63–66. DOI: `10.1109/ICMENS.2009.23`.

[31] A. H. Majeed, E. Alkaldy, M. S. Zainal, K. Navi, and D. Nor. "Optimal design of RAM cell using novel 2: 1 multiplexer in QCA technology." In: *Circuit world* (2019).

[32] A. H. Majeed, M. S. Zainal, and E. Alkaldy. "Quantum-dot Cellular Automata." In: *International Journal of Integrated Engineering* 11.8 (2019), pp. 143–158.

[33] 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), pp. 35–43.

[34] R. S. e. a. Mohammed Alharbi Gerard Edwards. "Novel Ultra-Energy-Efficient Reversible Designs of Sequential Logic Quantum-Dot Cellular Automata Flip-Flop Circuits." In: *PREPRINT (Version 1) available at Research Square [https://doi.org/10.21203/rs.3.rs-2145478/v1]* 1 (2022).

[35] 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), pp. 1233–1237.

[36] F. Pigorsch, C. Scholl, and S. Disch. "Advanced unbounded model checking based on AIGs, BDD sweeping, and quantifier scheduling." In: *2006 Formal Methods in Computer Aided Design*. IEEE. 2006, pp. 89–96.

[37] 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, pp. 2114–2117. DOI: 10.1109/ISCAS.2016.7538997.

[38] J. I. Reshi, M. T. Banday, and F. A. Khanday. "Sequential circuit design using quantum dot cellular automata (QCA)." In: *2015 Symposium on Computers, Communication and Electronic Engineering*. 2015, pp. 143–148.

[39] 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.

[40] R. R. Schaller. "Moore's law: past, present and future." In: *IEEE spectrum* 34.6 (1997), pp. 52–59.

[41] 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), pp. 1792–1795.

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

[43] G. Schulhof, K. Walus, and G. A. Jullien. "Simulation of Random Cell Displacements in QCA." In: *J. Emerg. Technol. Comput. Syst.* 3.1 (2007), 2–es. ISSN: 1550-4832. DOI: 10.1145/1229175.1229177.

[44] 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), pp. 147–159.

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

[46] 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), pp. 638–647.

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

[48] H. Thapliyal and N. Ranganathan. "Reversible logic-based concurrently testable latches for molecular QCA." In: *IEEE transactions on nanotechnology* 9.1 (2009), pp. 62–69.

[49] F. S. Torres, M. Walter, R. Wille, D. Große, and R. Drechsler. "Synchronization of clocked field-coupled circuits." In: *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)*. IEEE. 2018, pp. 1–5.

[50] G. Toth and C. S. Lent. "Quasiadiabatic switching for metal-island quantum-dot cellular automata." In: *Journal of Applied Physics* 85.5 (1999), pp. 2977–2984.

[51] 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: *2016 29th symposium on integrated circuits and systems design (SBCCI)*. IEEE. 2016, pp. 1–6.

[52] 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), pp. 34–44.

[53] M. Walter and R. Drechsler. "Design Automation for Field-Coupled Nanotechnologies." In: July 2020, pp. 176–181. DOI: `10.1109/ISVLSI49217.2020.00040`.

[54] 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). ISSN: 1550-4832. DOI: `10.1145/3312661`.

[55] M. Walter, R. Wille, F. S. Torres, D. Große, and R. Drechsler. "Scalable design for field-coupled nanocomputing circuits." In: *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 2019, pp. 197–202.

[56] K Walus, A Vetteth, G. Jullien, and V. Dimitrov. "RAM design using quantum-dot cellular automata." In: *Nanotechnology conference*. Vol. 2. 2003, pp. 160–163.

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

[58] 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, p. 143.