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

2.1 Representation of Logic Circuits

The Boolean Algebra, formed by mathematician George Boole in 1847, shows that every digital circuit can be represented by logic functions, independent of their underlying technology. Until today his work is the foundation for defining and discussing about them.

2.1.1 Boolean Functions

A definition of the Boolean calculus was first provided by Edward V. Huntington in 1993. From the *set of independent postulates for the algebra of logic* and his own correction [17, 16], 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_{abs}. The algebra includes the binary function $+ : B_{abs} \times B_{abs} \to B_{abs}$ and the unary function $' : B_{abs} \to B_{abs}$.

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

The last of his postulates named after the inventor and is commonly known as *Huntington equation*. Also it exists an "universe" element $u \in B_{abs}$ for which holds:

$$\begin{gathered} \exists u' : a + u' = a \\ \exists u : a + a' = u. \end{gathered}$$

The most common Boolean algebra $\mathbb{B}$ is defined by the 6-tuple $(\mathbb{B}, \vee, \wedge, \neg, 0, 1)$, where $\vee$ and $\wedge$ are another denotations for the binary operands for disjunction $+$ and conjunction $\cdot$ in $\mathbb{B}$. The third function $\neg$ describes the unary negotiation function, which was former denoted as $'$. The set $\mathbb{B}$ holds two distinct elements $\{0, 1\}$, with $u = 1$ and $\neg u = 0$ respectively [14].

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 [29]:

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 functions number of arguments or arity. 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 the integer $m > 0$.

Nonetheless, every k-ary function can still be decomposed into a set of the common Boolean functions $(\vee, \wedge,)$. A common notation for Boolean functions are the *conjunctive normal form* (CNF) and *disjunctive normal form* (DNF), using literals.

Definition 2.1.3. A literal is either an atom a (positive literal) or the negation of an atom $\neg a$ (negative literal).

Definition 2.1.4. 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 [42]:

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

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 and *De Morgan's laws* following from the definitions in 2.1.1, it follows that any Boolean Algebra can be reduced to only two operands, e.g. conjunction $(\vee)$ 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 most of them, including e.g. truth tables or reduced sum of products, suffer from drawbacks like exponential representations of basic functions like the parity (XOR) function. Even if a reasonable representation exists for a given function, simple operations like forming the complementary could yield an exponential function representation. Logic networks overcome these restrictions and have proven to be very useful transforming a circuit described

by logic functions into a gate representation. This process is also referred to as *logic synthesis*. An approach for logic networks is given in [5], describing *function graphs*:

Definition 2.1.5 (Function graph)**.** A function graph is a rooted, directed graph with vertex set V containing two types of vertices. A *nonterminal* vertex v has as attributes an argument index $index(v) \in \{1, ..., n\}$, and two children $low(v), high(v) \in V$. A *terminal* vertex v has as attribute a value $value(v) \in \{0, 1\}$.

Furthermore, for any nonterminal vertex v, if $low(v)$ is also nonterminal, then we must have $index(v) < index(low(v))$. Similarly, if $high(v)$ is nonterminal, then we must have $index(v) < index(high(v))$.

Definition 2.1.6 (Function Graph Boolean Functions)**.** A function graph G having root vertex v denotes a function f_v 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 nonterminal vertex with $index(v) = i$, then f_v is the function $f_v(x_1, ..., x_n) = \bar{x}_i \cdot f_{low(v)}(x_1, ..., x_n) + x_i \cdot f_{high(v)}(x_1, ..., x_n)$.

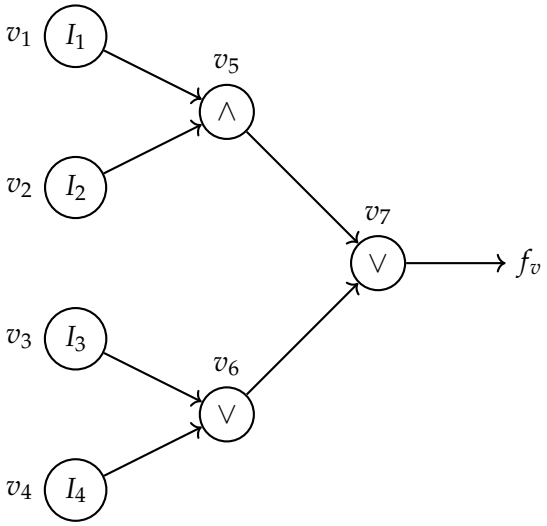
Since the definition reduces the number of children connected to a vertex to two, therefore only allowing binary Boolean Functions, a custom definition is given:

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 and outgoing edge $e_{out} \in E$ is called parent. V contains two types of vertices. A *nonterminal* 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 nonterminal vertex v, if $child_i(v)$ with $1 \leq i \leq l$, then we must have $index(child_i(v)) < index(v)$ respectively.

Definition 2.1.8 (Logic Network Boolean Functions)**.** A set of nary 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 $f_v = 1$



The corresponding recursive Boolean function reads:

$$f_v = f_v(v_7) \qquad f_v(v_6) = f_v(v_3) \wedge f_v(v_4)$$
$$f_v(v_7) = f_v(v_5) \vee f_v(v_6) \qquad f_v(v_5) = f_v(v_1) \vee f_v(v_2)$$

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

Figure 2.1: Binary Logic Network

b) If $value(v) = 0$, then $f_v = 0$

2. If v is a nonterminal 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 2.1.

Definition 2.1.9 (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 $\langle a, b, 1\rangle = a + b$.

Adapting the names used in the underlying libraries used to program the algorithms proposed subsequently following in this work (chapter 4), a terminal vertex is referred to as *primary input* (PI) with their set denoted as I. The set of nonterminal vertices referred to as *nodes* is denoted as Λ. From the definition follows $I \cap \Lambda = \emptyset$. An edge connecting a children v_i and 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 true. 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 widely used binary logic network is the *Majority-Inverter Graphs* (MIGs) utilizing the ternary majority function and negation. But there also exists a wide range of logic networks permitting more than just two node functions. One widely used example is the *XOR-AND-Inverter Graph* (XAG) with the parity function, conjunction and negation functions respectively.

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 in 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.

One drawback, that sticks to this definition of logic networks and holds also true for the aforementioned representations is that they are all *non-canonical*, which means that a given function can be represented by different logic networks. This 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 one Boolean Function, resulting in the widely accepted assumption, that the determination of an optimal logic network is a $\mathcal{NP}$-complete problem [38]. Attempts to create canonical logic networks, seem to evade this problem, but include co$\mathcal{NP}$-complete problems in itself [5]. Algorithms used for logic synthesis are therefore based on approximate solutions .

2.2 QCA Technology

Following the well known Moores law, CMOS technology is facing a multitude of challenges. Well-known are e.g. short channel effect, impurity variations, and most importantly the heat, resulting from static and dynamic power losses. To tackle these challenges the International Roadmap for Devices and Systems (IDRS), former ITRS, proposes solutions within the semiconductor domain, e.g. new materials and multi-core architectures. But also new technologies are 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, instead of voltage levels. Data between cells is transferred based on Coulomb repulsion, utilizing electromagnetic fields. This enables the technology to achieve high performance in terms of device density, clock frequency and power consumption [24].

2.2.1 Cells

As already mentioned, the elemental unit of this technology is a QCA-cell. Since there in no uniform way of build the quantum dots and connecting them to cells, we look at a rather lower-level abstraction depicted in figure 2.2. The four circles in the corners of the QCA-cells show quantum-dots, that can be implemented by any charge container with discrete electrical energy states. Further a cell contains two excess electrons, which can be localized by the quantum dots. The energy barriers or the quantum dots are able to trap the charge of the electron. If an electron is trapped inside a quantum-dot it

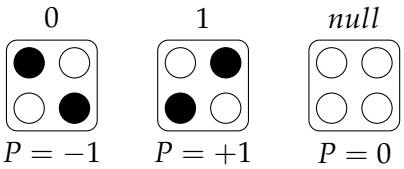


Figure 2.2: QCA-Cell sates

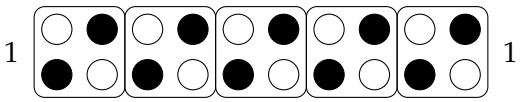


Figure 2.3: Adjacent QCA-cells forming a wire segment

is filled black. Due to the Coulomb repulsion the electrons occupy diagonally opposed quantum dots, resulting in two possible stable cell configurations and one unstable cell configuration [28, 20, 21].
A stable states indicates, that it is well distinguishable of the usual energy band. Therefore the energy difference between two consecutive energy states must be well above the thermal noise energy (k_BT). Only such states are suited for information transfer. The stable states can be derived from the 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 [28].
In order to transfer information, cells are placed side by side, whereby the polarization of the driver cell, which is the left most cell inputting the information, changes the polarization of the adjacent cell. When the adjacent cell is polarized it can give its state to the next cell and so on [20]. This simple structure is representing a wire in QCA-technology and its function is depicted in Figure 2.3.

2.2.2 Clocking

As already mentioned, the 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 right most cell, representing the output of the simplified QCA-circuit. The depicted state of the cell, where all neighboring cells have the same polarization is called ground state. Since the logic is deterministic, there exists exactly one ground state configuration of cell polarization for one choice of inputs. When in ground state, the circuit has minimum state energy.

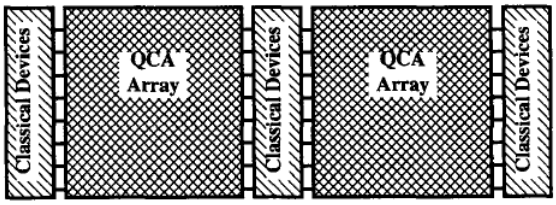


Figure 2.4: Schematic of a combined QCA and CMOS sytem [21].

Given a circuit in ground state, when the input of the system is changed, the systems energy rises as depicted in figure 2.5. It results from the so called *kink*-energy, which describes the energy difference between two QCA-cells with opposing polarization. Since the polarization of the cells changes one cell after the other, also the *kink* moves along the cells. After time τ the system dissipates again into the ground state, with newly computed outputs. While the kink-energy stays the same with an increasing number of cells, degeneracy of the excited states rises. This again means, that the system can be illegally in an excited state at non-zero temperature [21].

The described process is called abrupt switching with dissipative coupling to the environment and can be summarized in the following three steps [21]:

1. Write the input bit by fixing the polarization state of cells along the input edge.
2. Allow the array to relax to its ground state while the new inputs are kept fixed.
3. Non-invasively read the results of the computation by sensing the polarization state of cells.

In the second step, highly complex dissipation mechanisms like phonon and plasmon emission take place, making it nearly impossible to get a complete theoretical description of the system. The dissipation time τ is therefore determined via experiments.
The first and third step require an environment, which provides the features for fixing inputs and sensing the output polarization. Such a system has to be integrated with classical CMOS devices as seen in Figure 2.4.

In order to implement a full QCA-system without CMOS-components, the QCA-Array has to be divided into smaller decoupled sub-regions. Therefore an adiabatic switching or rather a clocking is introduced. The clocked regions are referred to as clocking zones. The clocking utilizes an external signal, the clock, to activate and deactivate said clocking zones. The approach first used, decreases the interdot barriers of all cells in a clocking zone, when applying a new input. When all cells in the region

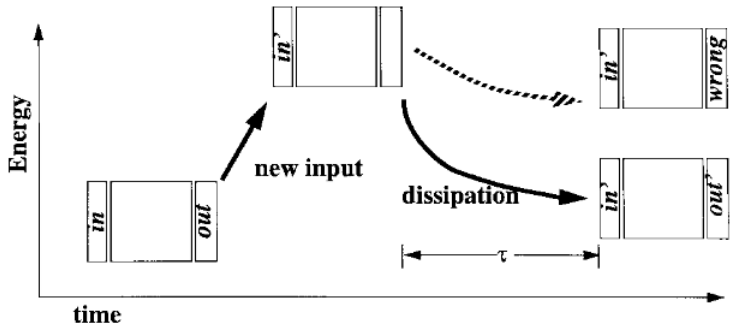


Figure 2.5: Schematic representation of a metastable state. Instead of relaxing correctly to the new ground state, a system may be delayed in an excited state due to an inability to tunnel through a kinetic barrier [20].

are stable, the barriers are raised again, while the barriers of the subsequent clocking zone are lowered simultaneously. This way the ground state gradually propagates through the whole circuit [20]. 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 [38].

A wire, divided in such clocking zones is shown in 2.3. 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 state is phase shifted by $\pi/2$, forming a 2π clock cycle. In the switch-phase cells start getting 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 [28]. The scheme of such a pipeline like clocking is depicted exemplary on a wire segment in Figure 2.6.

The described clocking is named *Landauer clocking*. The inventor Rolf Landauer himself pointed out the vast power dissipation of this clocking mechanism. This impairing property is already well known from the CMOS technology, resulting in several drawbacks including extreme cases of switched off chip areas referred to as dark silicon. One common approach in CMOS is the lowering of circuit frequency utilizing e.g. multi-chip architectures. To tackle this problem in the QCA domain, Landauer pointed out that the *erase* function has to be eliminated from the clocking. This function

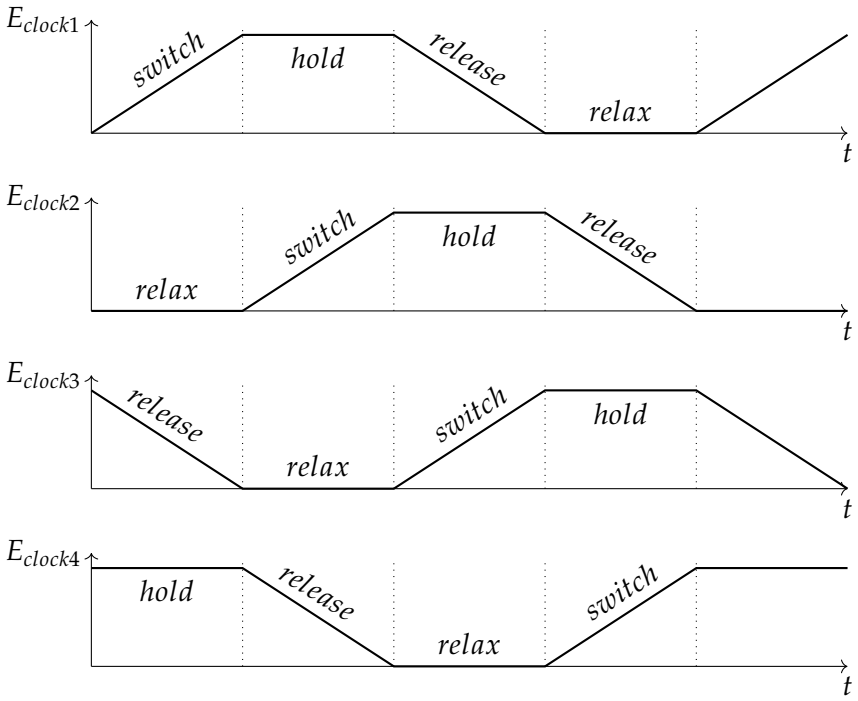
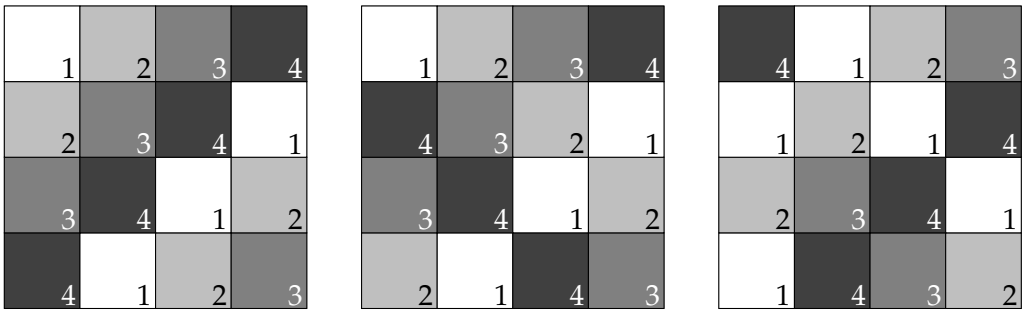


Figure 2.6: QCA clocking pipeline

is logically irreversible and describes the erasing of information in a clock zone, when passed to the next. He argues that every erased bit dissipates at least $k_B T \ln(2)$ in heat. Exemplary if a QCA-cell has size $1nm \times 1nm$ and operating frequency of $100GHz$, the corresponding density of devices results to $10^{14}\ cm^{-2}$. Further a dissipation of $0.1eV$ per electron every clock cycle is assumed, resulting in a total power dissipation of $160\ kW\ cm^{-2}$. This directly yields to the statement, that a device operating with this clocking would be inoperable (it would evaporate due to the heat) [19]. The *Bennet clocking* tackles exactly this problem by altering the timing of the clocking signals. Just as in the Landauer clocking the clocking-wave moves from left to right, 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 [19].
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 circuits with millions of cells, this clocking gets infeasible for large circuits. Also the scheme supports clocking of single cells, which means that electrodes of the same size have to be fabricated. Since this is also not feasible, this design is obsolete.
In order to attain uniform clocking zones with a possible distribution of clocking signals, the *tile-based* clocking is introduced. The approach of this design is to provide uniform square tiles of the size 3×3 or 5×5. For clocking tiles bigger than this, the information propagation was suggested to be erroneous, also following in an argument against cell-based clocking [33].

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 2.7 three clocking schemes are showed, each of them based on a different idea. Since information is only allowed in ascending clock order (except 4 to 1), the 2DDWave clocking scheme in figure 2.7(a) only allows information to propagate in two directions, south and north [37]. This simplicity allows no back propagation, prohibiting the placement of sequential circuits. Also it restricts gates in the scheme to have a maximum input size of two. The USE scheme 2.7(b) tackles the first problem by introducing clocking loops into the scheme, giving the possibility to place sequential circuits [7]. To tackle the second problem, the



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

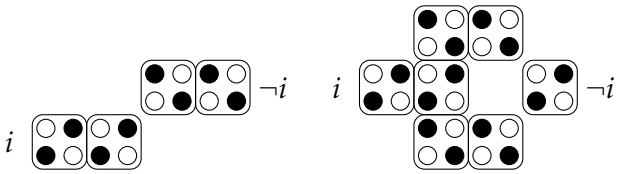
Figure 2.7: Different clocking Schemes in QCA

RES scheme 2.7(c) gives an opportunity to place gates of input size three. Since one tile is restricted to four adjacent cells of which one has to output the information of the cell, this gives the maximum input size allowed. This is especially important for the placement of majority-gates [13]. In QCA-technology they can be represented by only one tile, making it to a huge advantage over CMOS technology. This is further evaluated in the next subsection on gates.

2.2.3 Gates

In this chapter a library of gates is introduced, which are later used in the placement and routing algorithms. Some of these gates are inherited from the QCA ONE library [26], which is used as base for some works on placement and routing CITE. 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 [26], imposing the designer with even more restrictions. For this work, the standard cell library should only contain gates occupying one tile. Every other logic function is composed out of these standard cells. The tiles used are of the dimension 5×5, which means that all standard gates are reduced to this area.

The first gate in the library is the inverter or NOT gate. The simplest implementation is shown in figure 2.8(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 [28]. In order to get a more robust gate regarding disturbance, the C-shaped



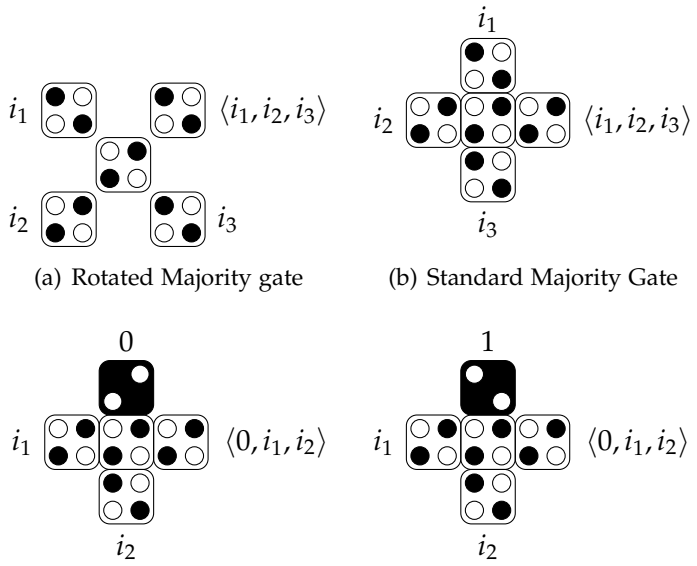
(a) 45° Inverter (b) Standard Inverter

Figure 2.8: Different QCA Inverter representations

inverter shown in figure 2.8(b) is introduced. This gate is used as standard in many libraries and works CITE but it still has to be mentioned that this implementation is really prone to complex single electron faults [23] and even common displacement faults [30], suggesting the addition of an inverter leg resulting to a E-shaped gate structure. Nevertheless the C-shaped inverter gate is part of the QCA ONE library and is also selected as standard cell for this work.

The next gate, which has to be investigated, is the majority gate, being the most important gate in QCA technology. Definition 2.1.9 suggests that the implementation of this function in CMOS technology requires multiple AND and OR gates to be placed and routed. In QCA technology on the other hand a majority function can be represented by exactly one gate, making it one of the major advantages over CMOS technology. There are two main implementations of the majority gate. The rotated majority gate in figure 2.9(a), which is used in QCA ONE, and the +-majority gate 2.9(b). Both of these implementations have their advantages and drawbacks. On the one hand the rotated majority gate exhibits sufficiently high degree of fault-tolerance against cell displacement or misalignment but has very poor degree of fault-tolerance against single cell omission or extra cell deposition [18]. The +-majority gate on the other hand is very prone to cell displacement but is also used as building block for AND and OR gates in most works. This means that the fabrication process for all these gates is very similar and since this work is more aimed to enhance the production of QCA circuits, the +-majority gate is chosen as standard gate for this work.
Following Definition 2.1.9 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 2.9(c).

Another major topic regarding gates are wires. Until now only straight planar wires have been introduced. In order to distribute information on the 2D grid, provided by the tile based layout, also bent wires have to be introduced. They are depicted in Figure



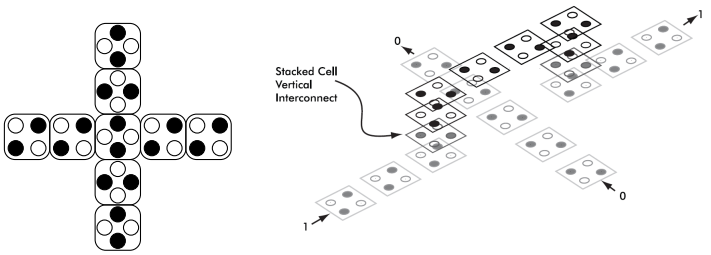
(a) Rotated Majority gate (b) Standard Majority Gate

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

Figure 2.9: The QCA Majority gate

2.11(f) and show a bend of 90 degrees. 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 multiply signals. This is done by adding a bent wire to a straight wire resulting in the fan-out shown in Figure 2.11(g).

The last special case of wires are crossing cases. By rotating the cells of one wire string by 45° the rotated cells don't have crosstalk with non-rotated cells [30] as shown in figure 2.10(a). This solution is very handy because it supports the planar structure of the circuit and is therefore called *coplanar crossover*. Further the possibility of multi-layer QCA has been investigated 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 2.10(b). The signal transmission in the vertical stacked cells works just as in horizontal direction. To impede any crosstalk between the wire strings two intermediate layers of cells are used in vertical direction. Theoretically the added layer cannot only be used as wire but since the signal distribution works just as in the ground layer also gates can be placed in these multi-layers. Simulations have shown that coplanar crossovers reduce the coupling between the horizontal string segments significantly. This makes the horizontal interconnect very sensitive to crosstalk and therefore highly prone to cell displacements. Multi-layer circuits on the other hand



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

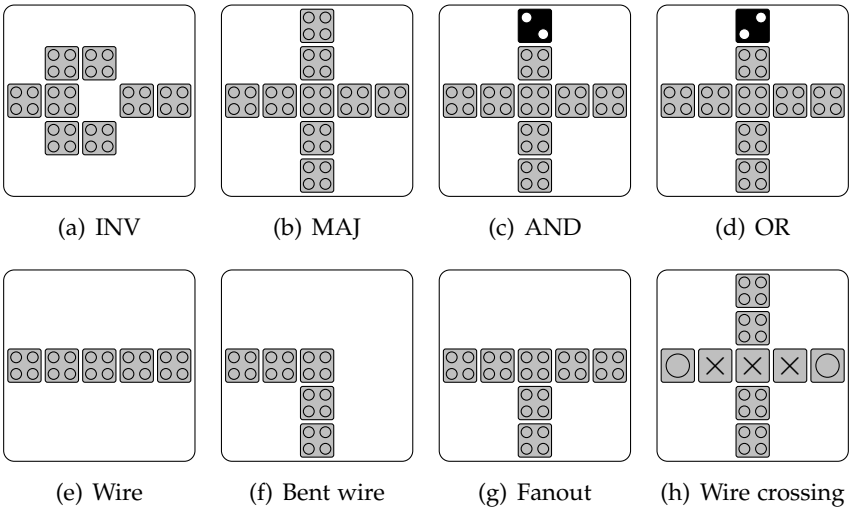
Figure 2.10: Different wire QCA wire crossing implementations

show a high robustness and therefore are used as standard in this library. In the gate 5×5 representation of the multi-layer interconnect, the raised wire string is described with a $\times$, while the vertical layers are described with a circle. In Figure 2.11 all gates used in this work are summarized.

Latches and Flip-Flops

Looking at a sequential circuit, it is the marriage of a combinational logic part and registers. When treating the combinational logic as blackbox, the outputs of it are storing the newly computed information into the registers and in the next cycle this information can be reused by connecting the registers again to the inputs of the combinational logic. This introduces a back-loop property, describing a signal propagating through combinational logic before getting stored and stabilized by a register and being reused again. This back-looping represents the key property of sequential circuits. Assuming that cominational logic can be implemented in QCA, registers or rather storage elements in general leave to be discussed. As already mentioned their task is to stabilize and store information which is then looped back via wires. To get a better understanding of how they can be implemented in QCA, first their main characteristics have to be derived from the well-known CMOS technology. Due to the already shown basic differences in signal propagation and clocking between the two technology domains, it will become clear that the implementation of latches and registers also has to be rethought from scratch.

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

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

Figure 2.11: QCA Standard Library

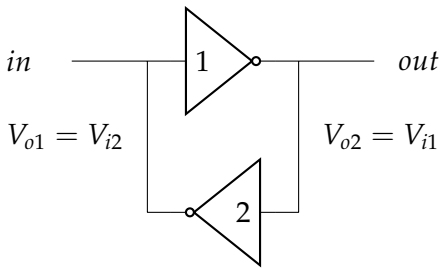
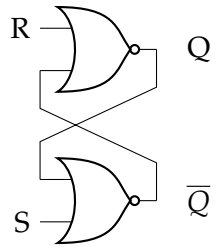


Figure 2.12: Eccles-Jordan-Flip-Flop



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

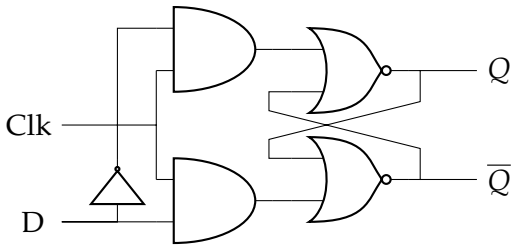
Figure 2.13: SR-Latch

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 [15]. To circumvent 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 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).

Because in QCA we only discuss synchronous logic, we also need to make this latch synchronous by clocking the Set and Reset inputs 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-latch is propagating the D input value, 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 it in the next clock cycle from its output. Also the master-slave principle eliminates the transparency. Regarding the term edge-sensitive for D-FFs, latches are



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

Figure 2.14: D-Latch

considered to be level-sensitive [15].
The goal of a storage element in QCA is to have all the properties of a D-FF, so it can store data from one clock cycle and pass it to the combinational logic in the next clock cycle. Also the element should not be transparent and the effect of an edge triggered element has to be discussed. In the QCA ONE library an effort was made to translate the D-FF into QCA by just replacing the CMOS gates and wires with the corresponding QCA gates. Thus a second external *clk* signal is introduced, mimicking the clock of a CMOS circuit. But since the QCA circuit has already an own clock with a dependency to all gates of the circuit and the clocking has some major differences from QCA to CMOS, as already observed in 2.2.2, this implementation is questionable.

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 [38]. 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 2.15 the clocking of such a 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 [38] 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

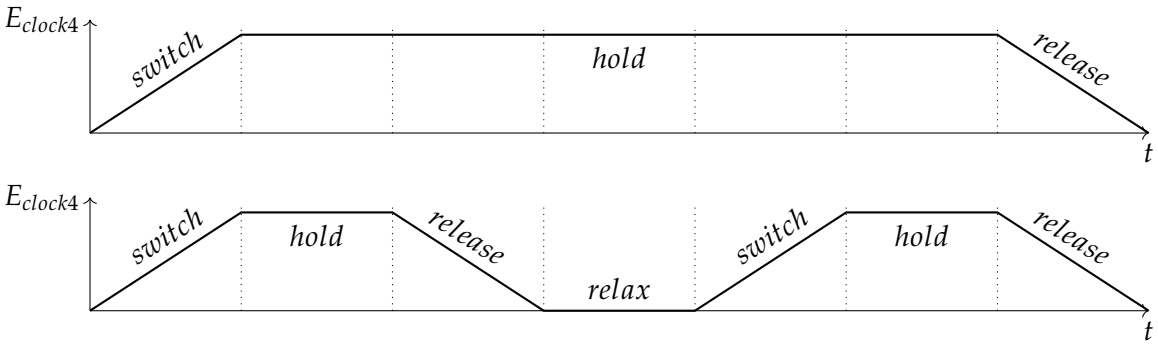


Figure 2.15: Clocking of a basic latch in QCA

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.

Further the idea of a wire FF should be discussed in the domain of placement and routing and not only as single element. First of all it can be examined, that the proposed FF implementation requires more complex clock generators for every latch and it is not sure if this is possible to implement. Also if we look back at the analogy of CMOS have now a combinational logic block and our storage element formed by a wire FF. But there is still a big difference. While in CMOS the information can just be arbitrarily be 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. 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. The idea, which is proposed in this work is now to utilize the delay, which is naturally occurring due to sequential wiring in order to mimic storage elements.
Figure 2.16 shows a schematic of a sequential circuit with a combinational logic and the wiring resulting from the need for a back-loop. At the output of the combinational logic there are the normal primary outputs and also register inputs. The register inputs are then wired to the corresponding register outputs, which are treated as primary input of the next cycle of the circuit. Because the information needs additional clock cycles to pass through the register, the throughput of the logic is slowed down. But since the loop has to be closed this slow-down cannot be prevented in QCA following this approach.

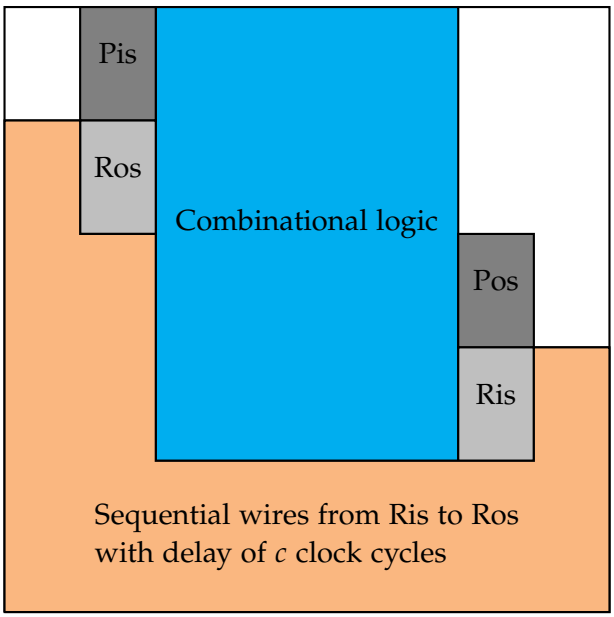


Figure 2.16: Scheme of a simple sequential circuit layout after placement and routing

2.3 P&R problem

As seen in the last sections of this chapter, placement and routing are strongly related to the clocking inside the QCA domain. In this section the denotation and constraints of placement and routing in the QCA domain are introduced. The P&R problem 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. Every *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 $N = (\Lambda, I, \Sigma, O)$. Besides 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 is able to 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 the 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. Further the *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 to be $\mathcal{NP}$-hard [39].
After reaching a gate-level layout still a technology has to be mapped onto it. For this work the in subsection 2.2.3 proposed standard library is used for the mapping. Although the definitions in this work are held quite generic because they are based on the book [38], defining the P&R problem for the field-coupled nanotechnologies domain. 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.

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. First the use both algorithms determining *optimal* and those who determine *scalable* solutions are explained. 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 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, a solution for the original problem can be derived from the solution of the propositional logic. The second approach *lazy SMT-solving* refers to the assisting use of *theory solvers* and 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 [2]. The abstraction is then passed to the SAT-solver, which again computes solutions and gives them to a set of theory solvers. They again check if the Boolean predicates hold true thus if they are consistent in the provided solution. If so the abstraction is satisfiability and the theory solver

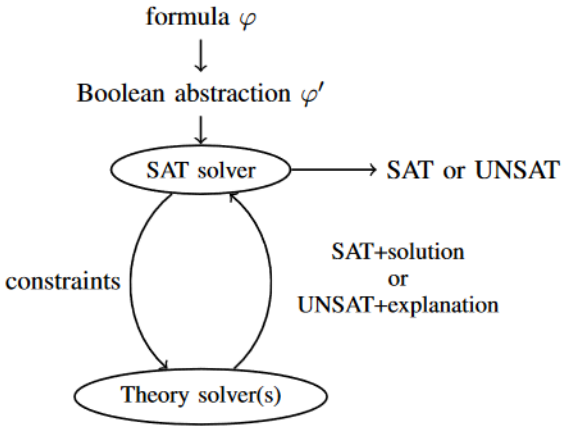


Figure 3.1: Lazy SMT-solving process [1]

instance returns SAT. Otherwise UNSAT with an explanation is passed back to the SAT-solver so that the abstraction can be improved. If the abstraction is found to be unsatisfied the problem is said to be unsatisfiable. [1]

With this knowledge optimal placement and routing algorithms can be discussed. Two approaches from [38] are reviewed for this. The first algorithm "Exact Placement and Routing" finds a valid placement, routing and clocking, also described as (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 optimal solution. The experimental results show that the determined layouts of the algorithm are many times smaller than the compared state of the art CITE. 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.

In the book [38] another exact P&R algorithm is proposed. The idea is to create a *one-pass synthesis*, which combines logic synthesis and physical design in a single run. 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 results lead to the usage of scalable placement and routing algorithms. These approaches trade optimality of the circuit, like layout size for computing 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.
Starting at the logic synthesis many works present a preprocessing of logic networks to make them suitable to translate them 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. Also the fanouts of the nodes are reduced, leading to a reduction of the place and rout complexity. One simple algorithm for this is to visit every node iteratively from the primary outputs 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. But this means that not only one node is duplicated but also all the nodes included in te sub tree rooted by it [34]. From this simple example we can already suggest that the insertion of dummy nodes can lead to uncontrollable growth of the logic network. Other algorithms which are not that dedicated, don't have such a high overhead in dummy nodes but therefore can't eliminate all wire crossings making it necessary to include nodes for crossings, so called *crossing edge insertions* [9]. Another preprocessing steps including the insertion of so called *buffer nodes* is the synchronization- Since the global timing constraint requires two paths leading to the same node to pass the same amount of tiles, this step makes sure that every two paths leading to a node include the same amount of gates. A buffer node therefore indicates that wires in QCA have to be viewed as gates as well. Also due to the insertion of buffers different partitions of a logic network can be generated [6]. Some approaches lead to even higher insertion of nodes indicating wasted area. This arises from the idea of a complete ternary logic network representation of a QCA circuit, since the gates are based on majority functions GRAPH . When extra area is produced due to the insertion of nodes, also wire lengths can be increased. For gates with two or three inputs, this means that if the longest wire has to be split into more than one clock zone, also the shortest wire has to be split into the same amount of clock zones in order to preserve the signal synchronization [34]. Another big problem of these algorithms is the requirement of cell-based clocking, which we have already showed to be insufficient. Even though there also exist algorithms using tile-based clocking they

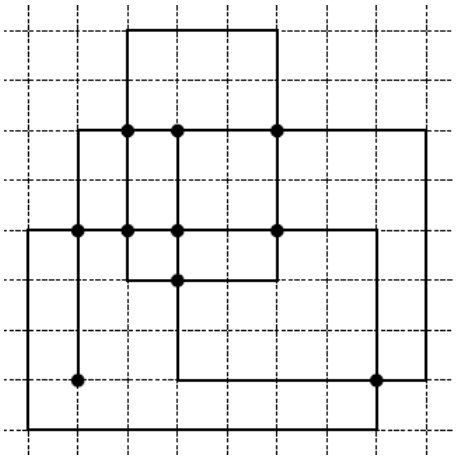


Figure 3.2: Example OGD drawing

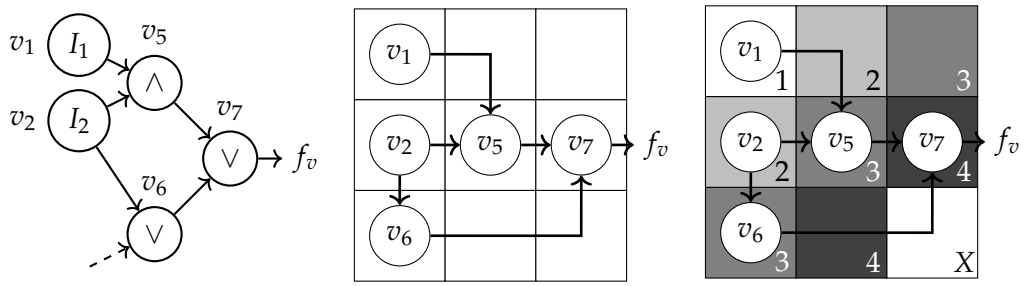
are limited by the general drawbacks of preprocessing [36] leading to exploding logic networks and even use greedy placement and routing algorithms limiting the approach to small and simple reconvergent patterns [11].

All these reasons lead to the proposal of *ortho*, an algorithm implementing a scalable placement, routing and clocking without preprocessing steps [40]. 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, have been proposed [6]. This means that the logic network is mapped 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 [10].

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 3.2 shows an example OGD. The dots in the graph represent vertices and are connected via straight line paths. Therefore the graph is drawn orthogonally.

Nonetheless an OGD also only respects the placement and routing, leaving the clocking to be addressed. The problem of insufficient clocking out of an OGD representation can be showed from the example in [38]. Figure 3.3 shows, that for a given OGD (3.3(b)) there has to be no clocking which can resolve the timing constraints (3.3(c)). It



(a) Partial representation of a logic network

(b) OGD representation of the logic network

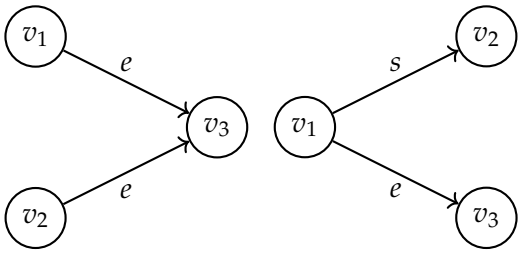
(c) No sufficient clocking for this OGD can be examined

Figure 3.3: Insufficient timing constraints of a OGD representation [38]

stands out that for the down right corner no clocking zone can be found that is satisfies the local synchronization constraint but also the global synchronization constraint is violated. Since the clocking or rather signal synchronization was a main task of the preprocessing now some other solution has to be found.

The idea used for the ortho algorithm comes from an extension to OGDs , which allows to determine a special OGD from a logic network in polynomial time being the constraint needed for a scalable approach. The base used here is formed by Therese Biedl [4], 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.



(a) Color assignment of incoming signals (b) Color assignment of outgoing signals

Figure 3.4: Relative positions of an OGD graph with correct color assinment

The relative position assignment under the proposed constraints can be seen at an arbitrary example in figure 3.4. In the example for outgoing edges (figure 3.4(a)), it is clear that the assignment constraint makes sure that two outgoing edges of the same vertex are routed without conflict. Due 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 3.4(b) depicts the assignment constraint for the incoming edges of one vertex. Since this implies only one color, also the layout only is extended in one direction, here in x-direction due to the east-coloring. The clocking used in the example is 2DD-Wave, because it also supports only signal flow in the directions *east* and *south*. This means that the local synchronization constraint is always true if we use this scheme. But due to its uniformity the scheme also maintains the global synchronization constraint for vertices placed after the proposed direction assignment. This is why ortho utilizes the 2DD-Wave scheme. However, there exist logic networks for which no coloring holding 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 the following the pseudo code of ortho, derived from [40] and depicted as algorithm 1, is 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. This again leads to a problem because inputs cannot wire to the south, so they need to be resolved by rewiring them each on a new column. With this input network the placing of the other vertices starts. For this two parameters need to be evaluated in each step. First the coloring of the node and second the updated parameter (w, h), saving the current dimensions of L. 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 nodes the node is wired to its predecessors. In case of *east* the wiring first goes east and the northern segment is then wired south. If the node is colored *south* the wiring goes south first and for the eastern child the segment then is colored east in order to connect to the placed node. 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 3.5 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

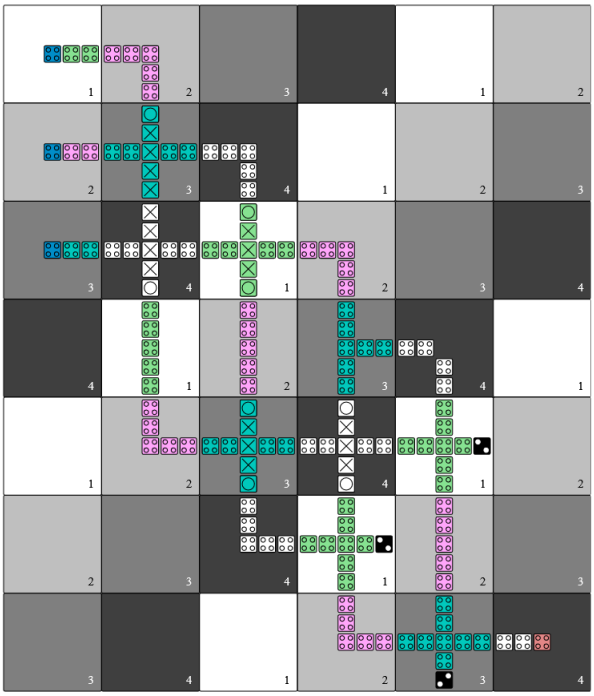


Figure 3.5: Placement and routing of a 2:1 mux network using the ortho algorithm

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 2.2.2, 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. Also back-loops are not allowed in the clocking, prohibiting the placement and routing of sequential circuits. Though the ortho algorithm provides an efficient tool for the placement and routing of purely combinational circuits , because as paper CITE has shown, the 2DD-Wave scheme is the most area efficient clocking scheme 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 [12] proposes an algorithm, which is based on [36]. As already discussed in the part about preprocessing, this algorithm brings some disadvantages, because dummy nodes are inserted as part of logic synthesis, what leads to an increase 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 chieves 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 wiring is used, so that some segments only have a distance of one tile and not like [41] suggests a minimum distance of two QCA cells. The use of these custom gates is because the algorithm used doesn't use the same constraints as ortho. The Ropper framework even routes wires above gates and doesn't use border inputs and outputs. On the basis of 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, which tries to implement majority gates for QCA is *migortho*, which is based on the ortho algorithm provided in fiction. The difference 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 this algorithm relying on the same argumentation is prone to cross.talk once more. 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 Sequential P&R Algorithms

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. Several attempts [22, 27, 31, 35] 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 [22]. The authors of [27] 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 2.2.2 and storage elements 2.2.3 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 [31] and reversible latches [35] it also is found that these works all use cell based clocking, which was already found to be insufficient in 2.2.2. The evolution to sequential tile based placement and routing was proposed through the USE clocking scheme [7], 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 [38]. The works [25] and [3] 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.
Another section of papers focus on the implementation RAM cells in QCA technology [41, 32]. Even though this paper focuses on a placement and routing algorithm for sequential logic, also the implementation of QCA storage could be adapted. An overview [8] over different RAM implementations shows that the state of the art is stuck at exactly this point. Without regards to the architecture of a QCA memory structure here a RAM cell according to the storage elements proposed in 2.2.3 is introduced. Figure 3.6 shows a RAM cell, with its respective wordline, bitline(s) and latch. The core of the cell is a latch, which is simply propagating the data in a circle and

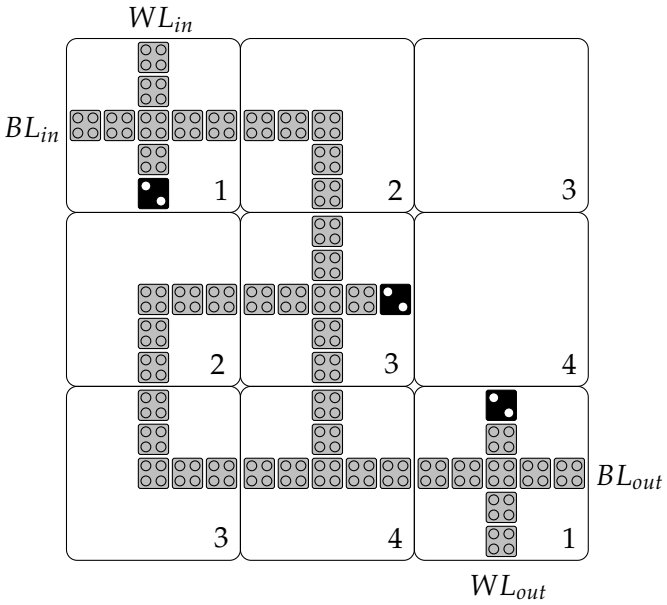


Figure 3.6: QCA RAM cell using a delay latch

therefore producing a stable output once in every clock cycle. The input mechanism works via two AND gates. The first AND gate is connected to the write-wordline, deciding if information from the bitline should be passed to the RAM cell. When the bit is passed the new state of the cell is computed by conjunction of the current bit and the newly inserted bit. Similarly for the read operation an AND gate representing the read-wordline needs input "1" in order to pass the data of the RAM cell to the outgoing bitline.

4 Methodology

This chapter proposes an algorithm, which is capable to overcome the restrictions of the ortho algorithm reviewed as state of the art in chapter 3. Therefore three different signal distribution networks are introduced. Namely an input, a majority and a sequential network. The name signal distribution networks comes from the resulting algorithm, which still lays combinational logic in the same fashion as the original ortho algorithm and only makes changes to the clocking scheme, when implementing irregular parts. Thus, distributing signals in a way that the new functionality can be implemented within the placement and routing procedure used by ortho. Further, because the 2DD-Wave scheme is used for the ortho algorithm, every majority gate and sequential circuit is seen as an irregularity or rather a signal distribution network that is put into the regular ortho scheme. In the following first the input network is discussed, which aims to reduce 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 Input Network

Looking again at the resulting layout of a 2:1 mux or ortho in 3.5, it can be seen that in the rows where inputs are placed, no other gates are put, because the space is used for rewiring in order to solve conflicts. The idea of the input network is to allow gates also to be placed in this area to save space and also place inputs in a way that wire crossings may be minimized. Recalling the pseudo-code 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 pseudo-code in 3 is implemented in a way that just finds *some* valid 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. One approach to solve this problem 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 input 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 in order to reduce wire expenses and crossings.

Algorithm 2 Ortho changes with input 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 : Δ → {east, south} and subdivide signals if necessary
Compute topological ordering v_1, ..., v_i ∈ N
Extend L by one column and reserve it for primary inputs
for all vertex v_1, ..., v_i ∈ 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(σ_1) = d(σ_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 ← 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 and the layout are part of the input network. If we look at the layout again the input network describes mostly the rows in which the inputs are placed, and whose area should be utilized. The viewed parts of the logic network are all primary inputs and the gates which they are wired to, skipping over inverters. This 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. One-input nodes including inverters and fan-out nodes can be

all wired east due to the fact that the primary input they are connected to only have one outgoing edge. This means that the coloring can be selected arbitrarily because no dependencies exist and in this case always the non-conflicting *east* assignment is 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. In every other case the direction assignment has to consider the coloring of the other incoming edge of the gate. Thus, first every primary input connected to a fan-out node and the fan-out node itself should be placed into the layout. This has several reasons. First of all the fan-out nodes give the constraints for the conditional coloring which is introduced in the input network. Also fan-out nodes produce excess wiring, which means that they should be connected to their outgoing gates as fast as possible. With the fan-out gates placed into the network we can now see that every fan-out gate needs exactly one output assigned with color *east* and one output assigned with color *south*. Considering that the gate connected to the edge colored *south* demands also his second incoming edge to be colored *south* and the second incoming edge could be connected to a primary input, we can see that a conditional coloring alone is not powerful enough to resolve all conflicts. For this case the coloring *south* is allowed in order to preserve the direction assignment rules but a new placement rule is introduced. The original algorithm part (line 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. Considering now the case of a *south* coloring a new rule can be deduced for a special case. If a node is labeled south and its predecessor, which has the lower horizontal position **also** has the higher vertical position, it is called the *root node* and the layout isn't extended by a column but the gate is still placed to 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 input 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 *south* colored primary input 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 southern of its predecessors. It was found that this rule could not only be utilized of this special case but also for the general *south* placement in the algorithm with one exception. Looking at fan-out nodes and 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, which is not allowed here. The resulting pseudo-code snippets replacing the used code are shown in !!. Also it has to be considered that this coloring still needs to include helping nodes e.g. when three fan-out nodes are connected to each other. Preceding to the coloring, the input nodes need to be ordered according to the ideas presented. Thus, primary

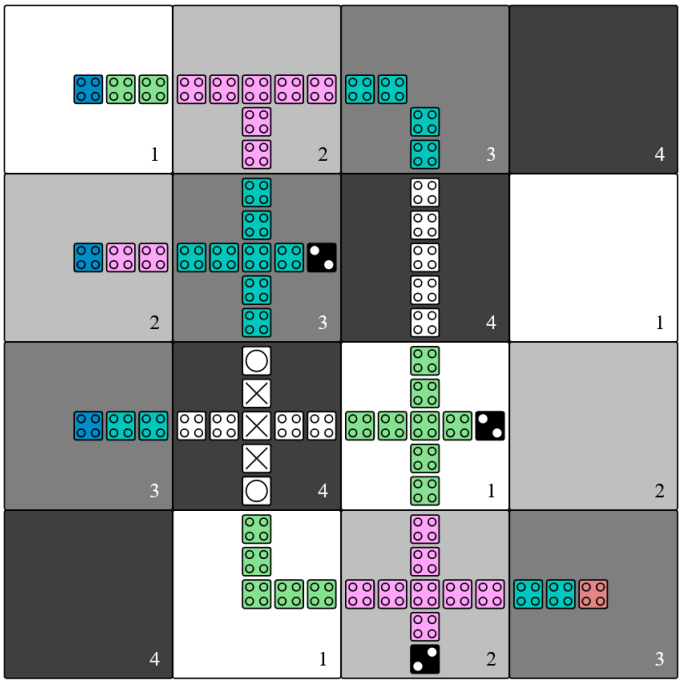


Figure 4.1: Placement and routing of a 2:1 mux network using the ortho algorithm with the input network

input nodes connected to fan-out nodes are placed first. 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 input network are placed arbitrarily. Some other issues are related to inverter nodes. As already mentioned they are skipped in the view of the input network. Considering 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, again a conflict arises because the input cannot wire to the east. 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 also a balancing network is introduced. Based on the substitution of the logic network into N with inverter and fan-out nodes it can happen that a fan-out node has two inverters connected to its outgoing edges. In this case these inverters are substituted again by one single inverter as incoming node to the fan-out, resulting in an overall lower number of inverter nodes.

Figure 4.1 shows the placement and routing of the ortho algorithm after implementing the proposed input network. It can be quickly seen that the resulting layout saves up place and even wire crossings. The ordering of the inputs puts first the fan-out node and then the two connected primary inputs. In this case the input 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. The exact results are shown in the next chapter.

4.2 Majority Gates Placement Network

Reference to the importance of this part in the QCA technology.
Point out that 2DD-Wave is still used but clocking scheme is adjusted to place majority gates.
This is why buffers have to be used.
Point out the overhead that is produced by buffers.
Maybe assumption why it doesn't make sense to only use RES. (you lose really much space for the cells which are no majority gates)

4.3 Sequential Circuits Placement

Importance of sequential circuits.
Point out how the registers are implemented.
Show how the distribution network is generated, where Ris and Ros are placed and how they are treated within the network.
Make clear that this implementation is slowing down the circuit significantly.

5 Experimental Evaluation

5.1 Benchmarks

For combinational and sequential

5.2 Results

Everything

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] 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.

[3] 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.

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

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

[6] 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.

[7] 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.

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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).

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.

[42] 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.