

1-1-2010

A Logic Formulation for the QCA Cell Arrangement Problem

Marc Stewart Orr
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Let us know how access to this document benefits you.

Recommended Citation

Orr, Marc Stewart, "A Logic Formulation for the QCA Cell Arrangement Problem" (2010). *Dissertations and Theses*. Paper 438.

<https://doi.org/10.15760/etd.438>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

A Logic Formulation for the QCA Cell Arrangement Problem

by

Marc Stewart Orr

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Xiaoyu Song, Chair
Douglas Hall
Jingke Li

Portland State University
©2010

Abstract

Some people believe that IC densities are approaching the fundamental limits inherent to semiconductor technologies. One alternative to semiconductors is Quantum-dot Cellular Automata (QCA); QCA is a nanotechnology that offers the potential to build denser IC's that switch at higher frequencies and run on lower power. QCA's most basic building block, the QCA cell, is inherently binary; digital circuits are implemented by arranging these QCA cells in pre-defined configurations on a two dimensional plane.

This paper proposes a logic formulation that describes arranging QCA cells on a two dimensional plane; it is presented as a set of rules that can be implemented with basic Boolean variables and operators. This Boolean formulation is general and can be applied to any given specification. In addition, an optimization constraint is defined so that the logic formulation will only validate the most efficient QCA cell arrangements. The correctness of the logic formulation has been empirically verified by testing it with a SAT solver. The effectiveness of the minimization constraint in conjunction with the logic formulation has been tested with a Pseudo-Boolean ILP solver.

Table of Contents

Abstract.....	i
Chapter 1 - Introduction	1
Chapter 2 - QCA Circuit Theory	5
2.1 <i>QCA Basics</i>	5
2.1.1 QCA Cell	5
2.1.2 Logic Gates	6
2.2 <i>QCA Interconnect Architectures</i>	7
Chapter 3 - Problem Description.....	11
3.1 <i>QCA Cell Arrangement Problem Definition</i>	12
3.2 <i>Model</i>	12
3.3 <i>The Proposed Solution</i>	13
Chapter 4 - Solving the QCA Cell Arrangement Problem	15
4.1 <i>Abstracting the QCA Interconnect – Virtual Edges</i>	15
4.2 <i>Achieving Connectivity for Multi-Terminal Nets</i>	17
4.2.1 Defining the Subnet Interconnect Problem.....	18
Chapter 5 - A Logic Formulation for 90 degree QCA Interconnect	19
5.1 <i>Two-Terminal Interconnect Constraints:</i>	19
5.1.1 Boolean Representation for QCA Two-Terminal Subnets	20
5.1.1.1 Valid Bits	20
5.1.1.2 Virtual Edges	22
5.1.2 Constraint 1: Terminal Valid Bit Constraint.....	24
5.1.2.1 Boolean Formulation for Constraint 1	24
5.1.2.2 Remarks on Constraint 1	24
5.1.2.3 Example of Constraint 1	24
5.1.3 Constraint 2: Terminal Arrangement Constraint	25
5.1.3.1 Boolean Formulation for Constraint 2.....	25
5.1.3.2 Remarks on Constraint 2	27
5.1.3.3 Example of Constraint 2	27
5.1.4 Constraint 3: Non-Terminal Arrangement Constraint	28
5.1.4.1 Boolean Formulation for Constraint 3.....	29
5.1.4.2 Remarks on Constraint 3	29

5.1.4.3	Example of Constraint 3	30
5.1.5	Theorem 1: Terminal Connectivity.....	31
5.1.5.1	Proof 1: Terminal Connectivity.....	31
5.1.6	Adjacent Cell Placement Problem	31
5.1.7	Constraint 4: Adjacent Cell Restriction Constraint	32
5.1.7.1	Boolean Formulation for Constraint 4.....	32
5.1.7.2	Remarks on Constraint 4	33
5.1.7.3	Example of Constraint 4.....	34
5.1.8	Relationship between Terminals and their Respective Nets.....	34
5.1.8.1	The Net ID Vector and its Boolean Representation	35
5.1.9	Constraint 5: Terminal Net ID Constraint	35
5.1.9.1	Remarks on Constraint 5	35
5.1.9.2	Example of Constraint 5.....	36
5.1.10	Constraint 6: Non-Terminal Net ID Constraint	36
5.1.10.1	Boolean Formulation for Constraint 6.....	36
5.1.10.2	Remarks on Constraint 6	37
5.1.10.3	Example of Constraint 6.....	37
5.2	<i>Constraints to Prevent Subnet Interference</i>	38
5.2.1	Adjacent Sub-Nets	38
5.2.2	Constraint 7: Adjacent Subnet Restriction.....	39
5.2.2.1	Boolean Formulation for Constraint 7.....	39
5.2.2.2	Remarks on Constraint 7	40
5.2.2.3	Example of Constraint 7	42
5.2.3	Multi-Terminal Re-Convergence.....	42
5.2.4	Constraint 8: Single Split Constraint	43
5.2.4.1	Boolean Formulation for Constraint 8.....	43
5.2.4.2	Remarks on Constraint 8	44
5.2.4.3	Example of Constraint 8.....	44
5.3	<i>QCA Polarity</i>	45
5.3.1	Constraint 9: Polarity Observance	45
5.3.1.1	Boolean Formulation for Constraint 9.....	46
5.3.1.2	Remarks on Constraint 9	46
5.3.1.3	Example of Constraint 9.....	46
5.4	<i>Minimization Constraint</i>	47
5.4.1	Random Wire Loops	47
5.4.2	Inefficient Cell Arrangements.....	48
5.4.3	Eliminating Random Wire Loops and Inefficient Cell Arrangements	49
5.4.3.1	Constraint 10.....	49

5.4.3.2	Formulation for Constraint 10	49
5.4.3.3	Remarks on Constraint 10	50
5.5	<i>Specifying Keep-Outs</i>	50
Chapter 6 - Constraint Implementation		51
6.1	<i>SAT Based Solution</i>	51
6.2	<i>Pseudo-Boolean Based Solution</i>	52
6.3	<i>Boolean to CNF Compiler</i>	54
6.3.1	Tseitin's Transformation for Basic Operators	54
6.3.1.1	Example 1	55
6.3.2	Converting Higher Level Operations into CNF Format	56
6.3.3	Implementation of the Boolean to CNF Compiler.....	57
6.3.4	The Structure of a Parse Tree.....	57
6.3.5	Boolean Expression Syntax	59
6.3.6	Building a Parse tree from a Boolean Expression	60
6.3.7	Building a CNF Expression from a Parse Tree.....	61
6.3.7.1	Support for Variables.....	62
Chapter 7 - Experimental Results		64
7.1	<i>System Specifications</i>	64
7.2	<i>Satisfiability and Pseudo-Boolean ILP Performance Results</i>	64
7.2.1	Runtime vs. The Number of Nets Specified	65
7.2.2	Runtime vs. T	66
7.2.3	Runtime vs. Grid Size.....	67
Chapter 8 - Conclusions		69
References:.....		71

Chapter 1 - Introduction

According to the International Technology Roadmap for Semiconductors (ITRS), fabrication facilities will be able to manufacture 20nm transistors in 2014; beyond this there is no proven solution that can be applied to fabricating smaller semiconductor devices [9]. Whether Moore's law breaks in 2014 or at a later date most people agree that it cannot hold true forever; there must be some physical limit restricting the number of semiconductor devices that can fit within a fixed area. Fortunately there are nanotechnology research activities focusing on new materials and computation paradigms that are fundamentally capable of providing smaller devices.

One promising nanotechnology is Quantum-dot cellular automata (QCA). Lent et al. pioneered quantum-dot cellular automata by introducing quantum-dot cells [4]. QCA cells can be used to build high density IC's that run on very low power and switch at teraflop frequencies [5][6]; this is achieved by combining them to form logic gates and wires [1][10][11]. Chapter 2 delves into QCA circuit theory by discussing its background and the specific QCA circuit structures used to build logic gates and an interconnect. QCA implements binary computation by placing logic gates and wires, in the form of QCA cells, on a 2-dimensional surface to perform specific logic functions.

Building a QCA IC requires designing a digital circuit down to the logic gate level and creating a cell arrangement to represent that circuit. Designing the gate level circuit can be done using the same methods that are currently applied to semiconductor based IC's. A cell arrangement is achieved by mapping the circuit's logic gates to their equivalent

QCA cell structures and placing the cell structures on a 2-dimensional surface. After the QCA logic gates are placed additional QCA cells must be arranged on the 2-dimensional surface to form an interconnect between all of the logic gates and IO's.

Previous work that maps logic gates to QCA cells and places them explored implementing sequential algorithms based on bipartite graph data structures [2] and [5]. Sequential algorithms have also been developed to arrange the QCA cells forming the global interconnect by representing it as a dynamic Vertical Constraint Graph [16]. Because QCA wire crossings are difficult to manufacture, all previous work makes an effort to minimize them.

The goal of this research project is to create a logic formulation for the QCA cell arrangement problem. More specifically the scope of the project is limited to accurately describing the 90 degree QCA interconnect with Boolean constraints. A logic formulation has scientific value because it clearly and specifically defines the QCA cell arrangements that are valid. In addition a number of frameworks like SAT solvers, LP solvers, SMT solvers, and assertion based verification tools require Boolean formatted specifications; these tools are useful for performing partial to full verification or even computing actual solutions. Understanding, designing, and verifying the QCA interconnect is of paramount importance because nanotechnologies are typically dominated by their interconnects [25][27].

Hung et al. developed a logic formulation for the traditional 3-dimensional routing problem that serves as the initial inspiration for the logic formulation explored in this paper [8]. Chapter 4 discusses the relevant theories derived in reference [8] and how they are applied to the QCA cell arrangement problem; section 4.1 develops a logical abstraction to represent the QCA interconnect that allows these ideas to be implemented. A final logic formulation is reached by formulating additional logic constraints to restrict the valid QCA cell arrangements to the interconnect structures defined in section 2.2. The QCA cell arrangement logic formulation is defined in its entirety in Chapter 5.

Chapter 5 also defines an optimization rule that improves the quality of valid QCA cell arrangements accepted by the logic formulation. This idea was motivated by Sagahyoon and Aloul's work done for CMOS Power Estimation [13]. They created a logic formulation and corresponding minimization constraint that could then be fed to a pseudo-Boolean ILP solver to determine the logic vectors that would result in maximum dynamic power dissipation for a CMOS IC.

Chapter 6 details the empirical test methodology employed to verify the logic formulation. The constraints developed in Chapter 5 are converted to an equivalent Boolean representation which is then compiled into CNF format using a Boolean to CNF compiler developed specifically for this project. The CNF representation of the logic formulation is fed into a SAT solver which returns a cell arrangement conforming to the logic formulation. The effectiveness of the minimization constraint in conjunction with the logic formulation is tested with a pseudo-Boolean ILP solver.

Chapter 7 presents experimental results obtained by applying the logic formulation and optimization rule to custom benchmarks with both a SAT solver and a pseudo-Boolean ILP solver; in the process, the feasibility of using these solvers to generate cell arrangements that conform to the logic formulation is tested. It is ultimately determined that using the logic formulation in conjunction with current pseudo-Boolean ILP and SAT solvers is too slow to obtain valid QCA cell arrangements in a design environment.

Chapter 2 - QCA Circuit Theory

QCA is one of many possible nanotechnologies that can be used to implement next-generation IC's. One of its biggest advantages is its ability to implement the Von Neumann architecture; this is because QCA cells can be combined to emulate traditional logic gates and wires [27][28]. This means that computers can be designed to function in the same fashion that they have for the last 50 years. Published papers detail the designs for QCA adders and basic ALU's. Beckett and Jennings suggest that often overlooked benefits will be realized if a nanotechnology is able to support the functionality of current micro-processors and run legacy code [27]. This chapter details how to arrange QCA cells to function as basic logic gates and wires.

2.1 QCA Basics

2.1.1 QCA Cell

The basic building block for Quantum Cellular Automata (QCA) is the Quantum-Dot cell (QCA cell). A QCA cell has four quantum dots arranged in a square pattern. Each QCA cell contains exactly two electrons. Electrons are confined to the quantum dots; they can tunnel to neighboring quantum dots within a QCA cell. Coulomb repulsion between the two electrons causes them to occupy opposite corners of the QCA cell. There are two symmetrical QCA cell configurations where the electrons are in opposite corners; these two configurations are used to represent a binary 0 and 1 for digital computations [10].

There are two orientations for QCA cells: 90 degree and 45 degree. 90 degree cells are the standard orientation. A 45 degree cell is just a 90 degree cell rotated 45 degrees

counterclockwise [5]. Figure 1 illustrates the physical binary 1 and 0 states for both 90 degree and 45 degree QCA cells.

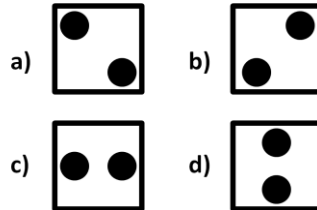


Figure 1: QCA cells. (a) A 90 degree QCA cell driven to a binary 0. (b) A 90 degree cell driven to a binary 1. (c) A 45 degree cell driven to a binary 0. (d) A 45 degree cell driven to a binary 1.

2.1.2 Logic Gates

Traditional logic gates are easily built from QCA cells. Figure 2 shows two different ways to build an inverter. Figure 2a illustrates a simple inverter. Because of Coulomb repulsion, the lowest energy state for two diagonally adjacent QCA cells is when they have opposite polarities. Figure 2b shows a second inverter configuration with the added benefit that the input is horizontally aligned with the output [10]. When diagonally adjacent QCA cells have opposite polarities all electrons occupy quantum dots separated as far away from each other as possible.

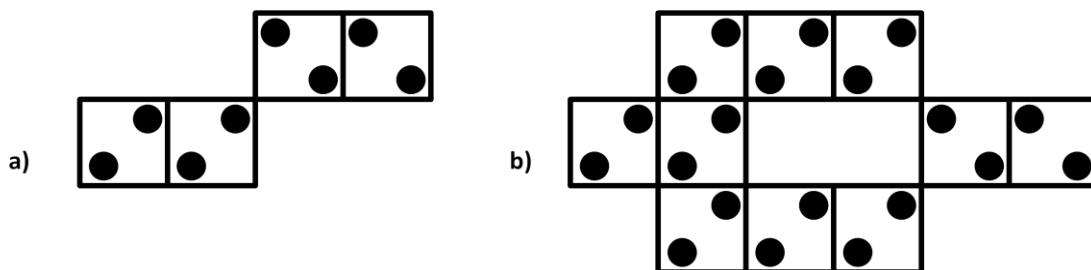


Figure 2: QCA inverter. (a) A simple QCA inverter. (b) A more complex QCA inverter; the advantage of this layout is that the output is horizontally aligned with the input.

The basic building block for QCA logic is the majority gate. Figure 3 shows the majority gate's schematic and truth table. When input A is forced to 0 the majority gate outputs the conjunction of B and C. The gate implements the disjunction of B and C when A is forced to 1. QCA circuitry can implement any combinational logic function with majority gates and inverters.

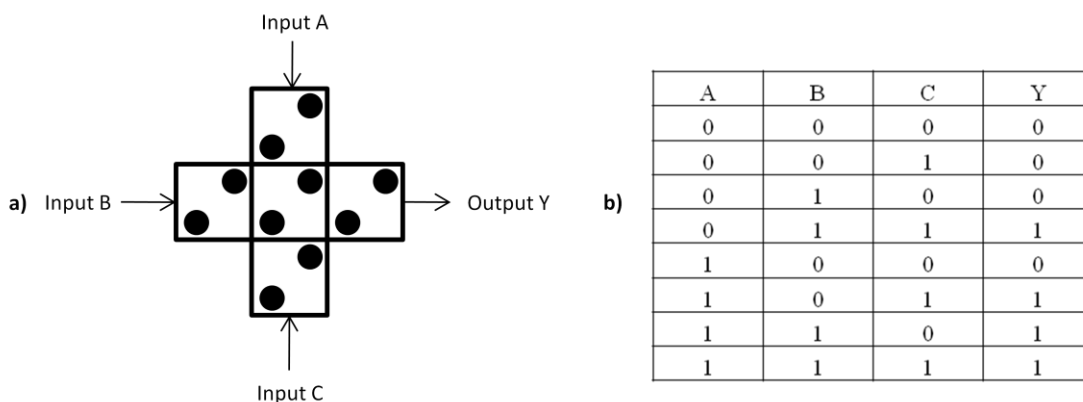


Figure 3: Majority Gate. (a) Schematic for a QCA majority gate. There are 3 inputs and 1 output. (b) Truth table for a QCA majority gate.

2.2 QCA Interconnect Architectures

Before deriving a solution to the QCA cell arrangement problem is important to develop an intimate understanding of QCA interconnect architectures. In other words, how does QCA circuitry transmit binary information from one physical location to another?

The most basic QCA interconnect architecture is the 90 degree QCA wire (illustrated in Figure 4a). It consists of one input and an arbitrary number of “free” cells. The input is driven to a binary one or zero by an external source; Coulomb interactions cause this value to propagate through the chain of “free” cells [1].

Figure 4b demonstrates an L-shaped 90 degree wire; this wire transmits its input value to all of its “free” cells. The shaded QCA cell in Figure 4b has 3 neighbors; the Coulomb forces from the 2 horizontal cells dominate. For L-shaped wires horizontal and vertical forces always dominate diagonal forces [1].

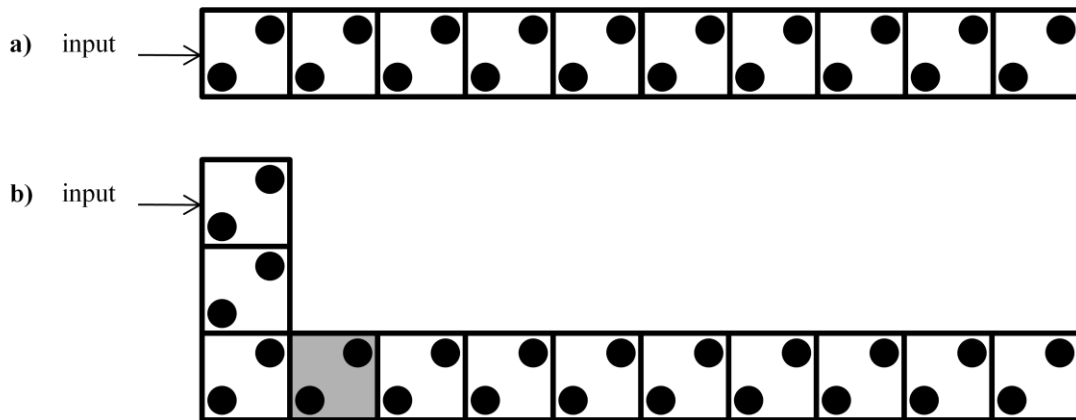


Figure 4: 90 degree QCA wires. (a) A straight 90 degree QCA wire. In this example the input is driven to a 1; this value then propagates along the contiguous chain of previously “free” QCA cells. (b) An L-shaped 90 degree QCA wire. When the wire turns, horizontal and vertical Coulomb interactions dominate causing the input value to propagate along the entire wire.

90 degree wires can branch allowing them to transmit information to more than one endpoint [1][6][7][10]. Different 90 degree fan-out configurations are shown in Figure 5. Figure 5a shows a maximum fan-out of three; a copy of the input is transmitted to all 3 endpoints. Figure 5b shows an inverting fan-out. Inverting fan-outs and standard fan-outs can occur at the same branch point. The inverting fan-out is not discussed in the literature; for the sake of generality, the cell arrangement solution in this paper allows inverting fan-outs. The final logic formulation can easily be modified to only return cell arrangements with standard “L-shaped” fan-outs.

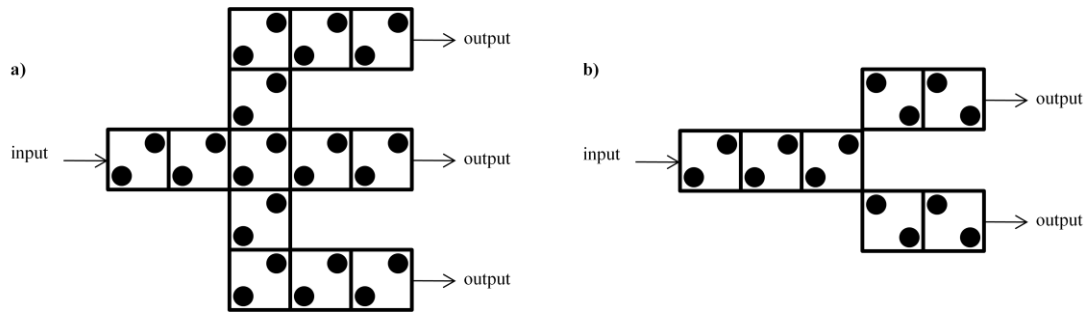


Figure 5: 90 degree wire fan-out. (a) An example standard fan-out with the maximum 3 branches. (b) An example inverting fan-out with 2 branches.

Signal propagation can also be achieved with chains of 45 degree QCA cells (illustrated in Figure 6). The 45 degree QCA wire consists of 1 input and an arbitrary number of “free” cells. A 45 degree QCA wire is sometimes called an “inverter chain” because adjacent cells have opposite binary values; the input is easily determined as long as the length of the wire is known [1].

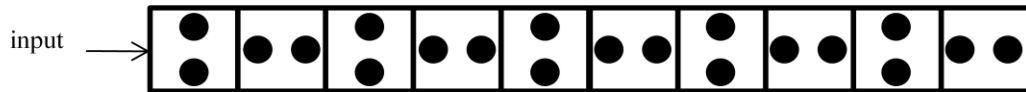


Figure 6: A 45 degree QCA wire; the input to this wire is a binary 1.

90 degree wires can be converted to 45 degree wires using a 3 cell construct called a “X-to-+ converter”; similarly 45 degree wires can be converted to 90 degree wires using a “+to-X converter” [1]. This 3 cell device is illustrated in Figure 7. The X symbol represents a 90 degree QCA cell, which houses its quantum dots in its 4 corners. The + symbol represents a 45 degree cell, which houses its quantum dots along the centers of its 4 walls.

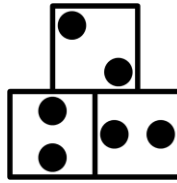


Figure 7: This picture illustrates an “X-to-+ converter” or “+to-X converter” depending on the direction of signal propagation. If the signal is being driven from the 90 degree cell to the 45 degree cells then it is a “X-to-+ converter”; otherwise it is a “+to-X converter”.

QCA technology allows 90 degree wires and 45 degree wires occupying the same plane to cross each other (this is illustrated in Figure 8) [1][2]. Wire crossing is especially useful because of manufacturing limitations that restrict designs to one layer; even so, it is desirable to minimize wire crossings because of difficulties associated with manufacturing 90 degree cells and 45 degree cells on the same die [3][26][28].

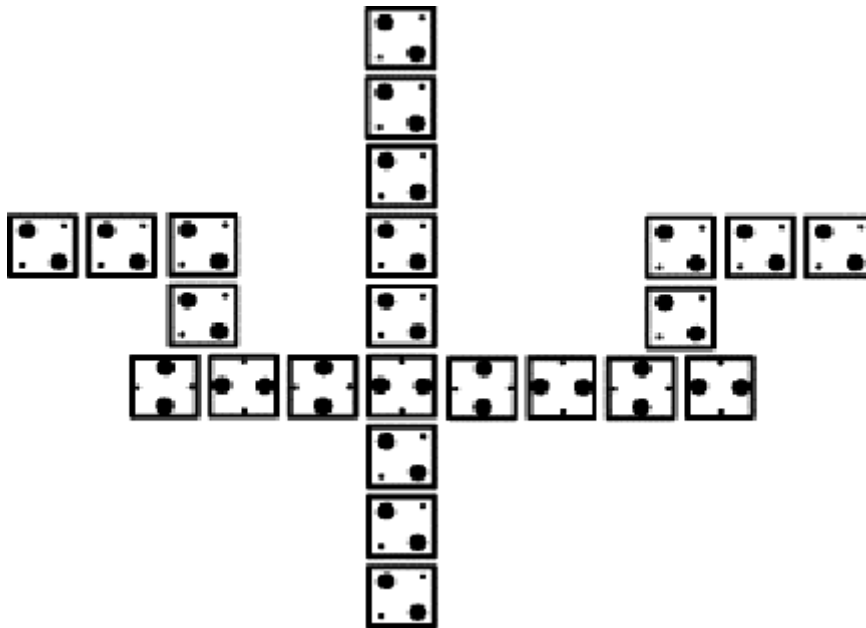


Figure 8: This figure was obtained from [19]. It illustrates coplanar wires crossing each other. The horizontal 90 degree wire, which is carrying a logic 0, is converted to a 45 degree wire (via a “X-to-+ converter”); it then crosses the vertical 90 degree wire and is converted back to a 90 degree wire (via a “+to-X converter”). The vertical wire is also carrying a logic 0.

Chapter 3 - Problem Description

Figure 9a shows an example QCA cell arrangement problem. In Figure 9a, a 4x8 QCA grid is specified ($W=4$, $L=8$). Two Sets of terminals are given ($N=2$). The first set of terminals (net 0), comprises 2 grid locations to be connected ($\hat{n}_0=2$). Net 1 comprises 3 terminals ($\hat{n}_1=3$). Figure 9b shows one valid solution to the problem specified in Figure 9a.

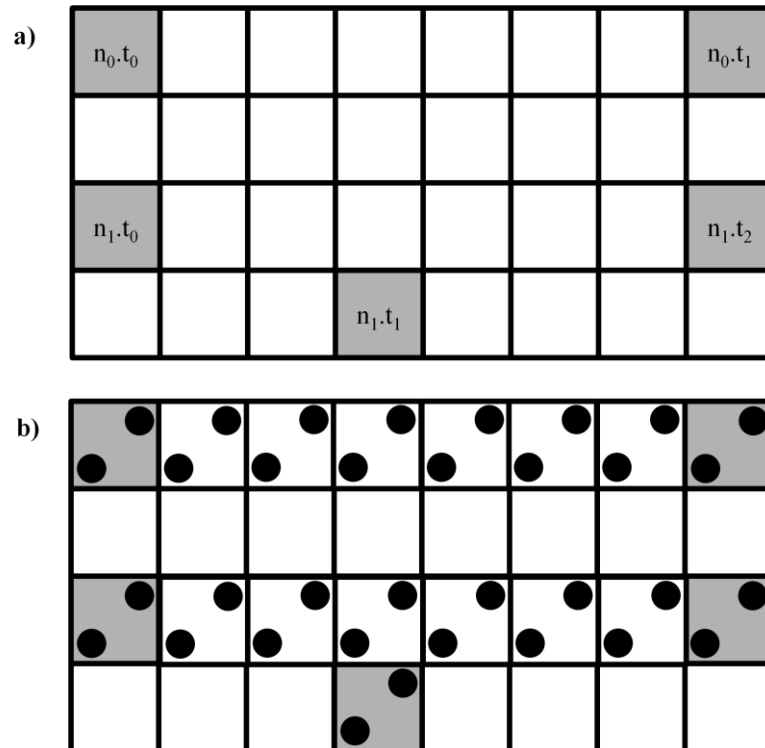


Figure 9: An example problem. (a) 2 nets must be completed on a 4x8 QCA grid. Net 0 has 2 terminals at locations (0, 0) and (0, 7). Net 1 has 3 terminals at locations (2, 0), (3, 3), and (2, 7). (b) One possible solution to the problem specified in part (a).

3.1 QCA Cell Arrangement Problem Definition

Given:

1. A QCA Grid, defined by its width and length (in terms of grid cells).
2. A set of terminals, defined by their QCA Grid locations and net assignments.

Arrange QCA cells within the QCA grid so that valid interconnects are formed between all of the terminals assigned to the same net; all nets must be fully connected.

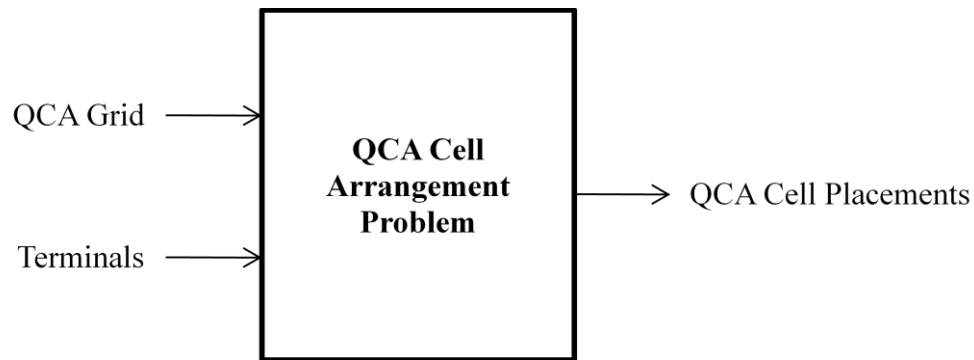


Figure 10: Block diagram for the QCA cell arrangement problem.

3.2 Model

The QCA grid comprises $W \times L$ cell locations and is modeled by $R = \{0, 1, \dots, W-1\} \times \{0, 1, \dots, L-1\}$. A set of terminals is a list of grid locations, $t_0, t_1, \dots, t_{\hat{n}}$, that are specified to be connected together. N sets of terminals, or nets, are given: n_0, n_1, \dots, n_{N-1} ; the number of terminals in each net is denoted by $\hat{n}_0, \hat{n}_1, \dots, \hat{n}_{N-1}$, respectively.

3.3 The Proposed Solution

A logic formulation for the QCA cell arrangement problem can be formulated by applying the basic ideas developed by Hung et al. to the virtual edge abstraction developed in section 4.1. Some additional constraints specific to the QCA cell arrangement problem are formulated as well. The additional constraints developed for the QCA cell arrangement problem result in “clean” cell arrangements by guaranteeing that subnets belonging to the same net do not interfere with each other. These constraints can be applied to the formulation created by Hung et al. to get higher quality 3-dimensional routings. The correctness of the final QCA cell arrangement logic formulation is verified by converting it to an equivalent Boolean representation for specific cell arrangement problems and using a SAT solver to create cell arrangements that can be verified.

To convert the Boolean formulation to its equivalent CNF representation a Boolean to CNF compiler was implemented. This tool accepts direct Boolean syntax. Other tools described in academic literature do not accept direct Boolean syntax [17]. In addition, the tool developed in this paper allows the user to define variables allowing compact formulations to be more easily developed and read. The author is not aware of any other CNF conversion tool that implements variable creation.

Work done by Sagahyoon and Aloul inspired the author to experiment with optimization constraints. An optimization constraint is implemented by converting the QCA cell arrangement formulation specified in CNF format to OPB format so it can be processed by a Pseudo-Boolean ILP solver. A higher quality cell arrangement can be realized by

adding a minimization constraint that clearly results in cleaner cell arrangements with minimum length interconnects.

Chapter 4 - Solving the QCA Cell Arrangement Problem

Chapter 5 defines precise QCA placement constraints necessary to achieve valid 90 degree QCA wires that connect a given set of nets defined by their terminals. This section describes the main theory behind these constraints.

4.1 Abstracting the QCA Interconnect – Virtual Edges

A QCA cell belonging to one of the interconnect architectures defined in Section 2.2 can be defined as either a terminal or a non-terminal. Terminal cells are the end points of any QCA interconnect. All other QCA cells are non-terminals.

For wires with two endpoints in traditional electronics frameworks, like the one defined in [8], terminals have exactly one connection and non-terminals have exactly 2 connections. QCA cells are no different; for interconnects comprising cells that have a maximum of 2 neighbors this is obvious. Figure 11 illustrates the connections between cells combined to form a straight 90 degree interconnect. These connections, referred to as virtual edges, have no physical representation in a real QCA layout; rather they are inferred by the spatial locality of adjacent QCA cells.

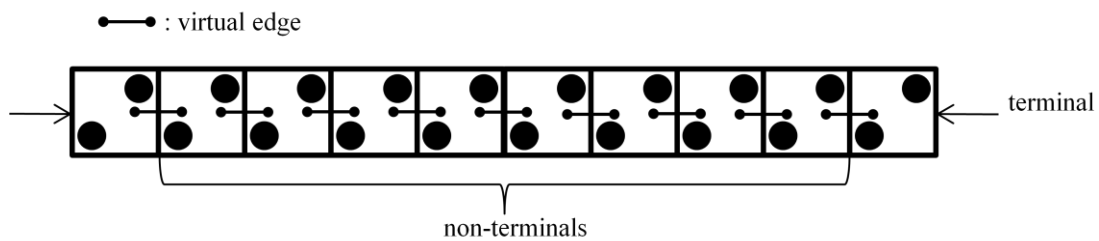


Figure 11: A 90 degree wire annotated with virtual edges.

For two terminal L-shaped interconnects things are not as obvious. The schematic for a two terminal L-shaped interconnect shown in Figure 12 illustrates the virtual edges. Even though the shaded cell has three neighbors it has exactly two virtual edges; this is because horizontal and vertical Coulomb forces dominate diagonal Coulomb forces [1]. In fact, every single cell in the L-shaped interconnect is attached to exactly two virtual edges except for the terminals which are both attached to exactly one virtual edge.

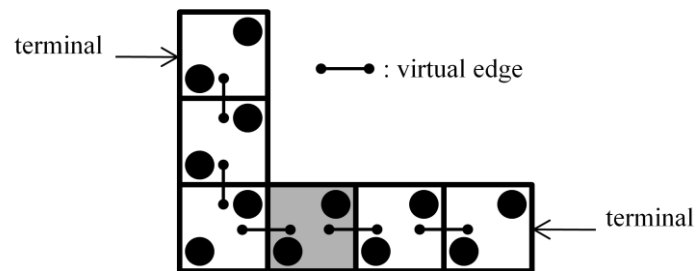


Figure 12: The virtual edges are drawn on this schematic of an L-shaped wire. The shaded cell has 3 neighbors but is only attached to 2 virtual edges. This is because the horizontal Coulomb forces dominate over the diagonal Coulomb force of the third QCA cell.

Figure 13 illustrates the virtual edges for an arbitrary two terminal inverting interconnect. This schematic is very straightforward like the 90 degree straight wire. All terminals are attached to exactly one virtual edge and all non-terminals are attached to exactly 2 virtual edges.

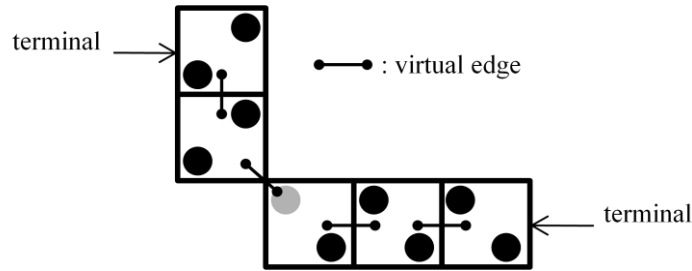


Figure 13: Virtual edges for an arbitrary inverting interconnect.

4.2 Achieving Connectivity for Multi-Terminal Nets

Reference [8] proves that a multi-terminal net can be connected by breaking it up into several two-terminal subnets. This strategy can be applied to quantum interconnects as well. The strategy works by choosing one terminal that every other terminal within the net is connected to. Formally, every multi-terminal net is divided into $\hat{n}-1$ two-terminal sub-nets:

$$s(n,1) = \{(x_0, y_0), (x_1, y_1)\}$$

$$s(n,2) = \{(x_0, y_0), (x_2, y_2)\}$$

...

$$s(n, \hat{n} - 1) = \{(x_0, y_0), (x_{\hat{n}-1}, y_{\hat{n}-1})\}$$

Figure 14 shows 5 terminals, all of which belong to the same net; each terminal is connected through $t_0 = (x_0, y_0)$. Notice that $s(n, 1)$ and $s(n, 2)$ converge with each other on their way to t_0 . Subnets are allowed to overlap because they are by definition apart of the same net. Different nets are not allowed to overlap as this would create a short circuit. This means that any grid cell can be assigned to two subnets as long as they belong to the same net.

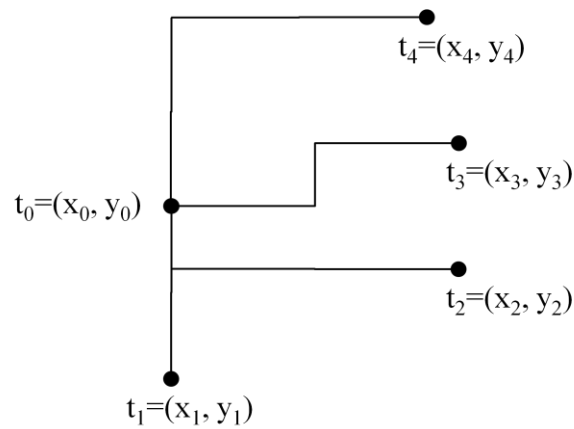


Figure 14: A multi-terminal net can be connected by dividing it into $\hat{n}-1$ subnets and forming interconnects for each subnet. This example shows 5 terminals divided into 4 subnets. Each subnet contains t_0 . Subnets can touch each other as long as they belong to the same net.

4.2.1 Defining the Subnet Interconnect Problem

A subnet comprises two terminal cells and all of the QCA cells connecting them. The subnet interconnect problem is: given the locations of two terminals, make a contiguous connection between them by placing QCA cells on the grid. Full connectivity for a multi-terminal net can be achieved by forming interconnects between all of its subnets.

Chapter 5 - A Logic Formulation for 90 degree QCA Interconnect

This chapter formulates Boolean constraints that describe the 90 degree QCA cell arrangement problem. These constraints can be tested by applying them to specific cell arrangement problems with a SAT solver or a Pseudo-Boolean ILP solver.

5.1 Two-Terminal Interconnect Constraints:

Multi-terminal QCA nets are connected by:

1. Breaking them into sets of two-terminal subnets (described in section 4.2).
2. Connecting all two-terminal subnets.
3. Guaranteeing that arranged subnets connect terminals belonging to the same net.

Different subnets may be assigned to the same grid locations if they belong to the same net. Subnets belonging to different nets cannot be overlapped or even placed adjacently.

These ideas are demonstrated in Figure 15. Figure 15a shows a 5x5 grid with 2 nets specified; these nets are summarized in Table 1.

Net ID	Terminals
Net 0	(0, 0); (0, 4); (2, 4)
Net 1	(4, 0); (4, 4)

Table 1: A summary of the nets and their terminals defined for the 5x5 grid in Figure 15a.

Net 0 comprises 3 terminals and Net 1 comprises 2 terminals. Net 0 is arbitrarily divided into 2 subnets:

$$s(n_0,1) = \{(0, 0), (0, 4)\}$$

$$s(n_0,2) = \{(0, 0), (2, 4)\}$$

Net 1 can be viewed as one subnet: $s(n_1, 1) = \{(4, 0), (4, 4)\}$. Figure 15b shows one potential solution determined by arranging all 3 subnets; several important observations are made. No grid locations have been assigned to both Net 0 and Net 1 (Net 0 and Net 1 never overlap); as a result only terminals belonging to the same net have been connected together. A few grid locations $\{(0, 0), (0, 1), (0, 2)\}$ have been assigned to both subnet $s(n_0, 1)$ and $s(n_0, 2)$ (these subnets overlap); this is allowed because these subnets belong to the same net.

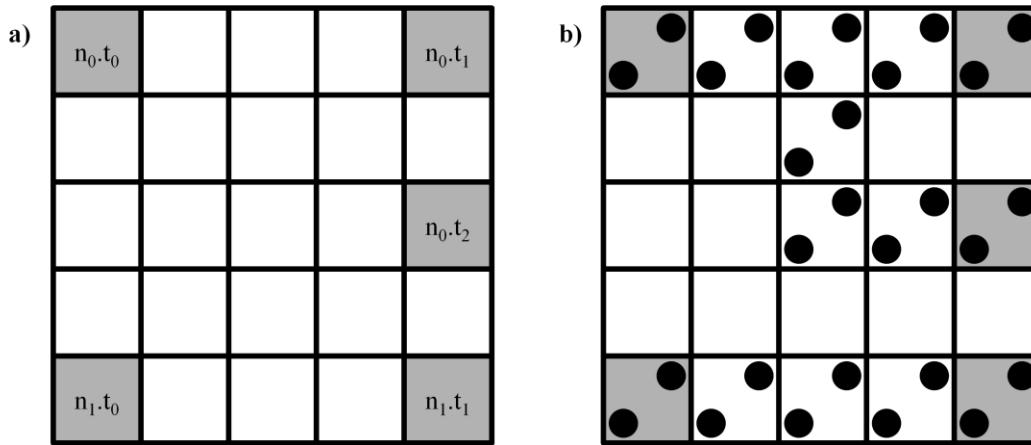


Figure 15: (a) Two nets are defined. Net 0 has 3 terminals (and 2 subnets). Net 1 has 2 terminals (and 1 subnet). (b) Valid cell arrangements for Net 0 and Net 1.

5.1.1 Boolean Representation for QCA Two-Terminal Subnets

5.1.1.1 Valid Bits

Let T be the maximum number of two-terminal subnets in any given net:

$$T = \max\{\hat{n}_0, \hat{n}_1, \dots, \hat{n}_{N-1}\} - 1$$

There can be at most T subnets assigned to any grid cell; this is because a grid cell cannot be assigned to multiple nets. For every grid cell location T valid bits are defined: $v_0, v_1,$

..., v_T . A subnet, $s(n,i)$, is assigned to a grid cell if that cell's valid bit, v_i , is asserted; in general a subnet, $s(n,i)$, is formed by contiguous grid locations that all have v_i asserted. A valid bit also implies that a QCA cell will be placed at that grid location in the cell arrangement.

Figure 16 shows the valid bits corresponding to Figure 15. The valid bit variables are first defined in Figure 16a; each grid cell is assigned 2 valid bits ($T = 2$). The name for each valid bit variable is appended with its location's coordinates to differentiate it from valid bits defined for other grid locations. Figure 16b shows the values assigned to these valid bits to achieve the cell arrangement in Figure 15b; the valid bits are arranged as bit vectors v_1v_0 . Grid locations with no valid bits asserted ($v_1v_0=00$) are left vacant; QCA cells are placed at all other grid locations. Any grid location that has both of its valid bits asserted is assigned to both subnets; these cells are no different physically than other cells assigned to just one subnet.

a)	$v_{0_0_0}$ $v_{1_0_0}$	$v_{0_0_1}$ $v_{1_0_1}$	$v_{0_0_2}$ $v_{1_0_2}$	$v_{0_0_3}$ $v_{1_0_3}$	$v_{0_0_4}$ $v_{1_0_4}$
	$v_{0_1_0}$ $v_{1_1_0}$	$v_{0_1_1}$ $v_{1_1_1}$	$v_{0_1_2}$ $v_{1_1_2}$	$v_{0_1_3}$ $v_{1_1_3}$	$v_{0_1_4}$ $v_{1_1_4}$
	$v_{0_2_0}$ $v_{1_2_0}$	$v_{0_2_1}$ $v_{1_2_1}$	$v_{0_2_2}$ $v_{1_2_2}$	$v_{0_2_3}$ $v_{1_2_3}$	$v_{0_2_4}$ $v_{1_2_4}$
	$v_{0_3_0}$ $v_{1_3_0}$	$v_{0_3_1}$ $v_{1_3_1}$	$v_{0_3_2}$ $v_{1_3_2}$	$v_{0_3_3}$ $v_{1_3_3}$	$v_{0_3_4}$ $v_{1_3_4}$
	$v_{0_4_0}$ $v_{1_4_0}$	$v_{0_4_1}$ $v_{1_4_1}$	$v_{0_4_2}$ $v_{1_4_2}$	$v_{0_4_3}$ $v_{1_4_3}$	$v_{0_4_4}$ $v_{1_4_4}$

b)	11	11	11	01	01
	00	00	10	00	00
	00	00	10	10	10
	00	00	00	00	00
	01	01	01	01	01

Figure 16: (a) Every grid cell has 2 valid bits defined ($T=2$) (b) The solution in Figure 15b redrawn to show the valid bits; they are communicated as bit vectors (v_1v_0).

5.1.1.2 Virtual Edges

For every pair of adjacent grid cell locations T virtual edge variables are defined: ve_0, ve_1, \dots, ve_T ; this results in T sets of virtual edges for each subnet to be arranged independently. A virtual edge, ve_i , is always associated with its respective subnet, $s(n,i)$ and is used to effect the assignment of its subnet's valid bits. If a virtual edge, ve_i , is asserted true then the two adjacent grid cells associated with ve_i both have their valid bits, v_i , asserted true; if that virtual edge, ve_i , is asserted false then the 2 adjacent grid cells cannot both have their valid bits, v_i , asserted true.

Figure 17a marks every adjacent grid cell location where a set of virtual edges must be defined for the example in Figure 15. Figure 17b in conjunction with Table 1 demonstrates how to name the virtual edges to differentiate between their location and what subnet they are associated with. Figure 17b labels a few of the virtual edge locations from the upper left hand corner of Figure 17a. Table 1 defines the virtual edge variables associated with some of these virtual edge locations for the example in Figure 15. The naming convention is to append to "ve" the valid bit associated with the virtual edge followed by that edge's 2 adjacent grid locations. Generally the coordinates for the left grid cell or top grid cell are written first; the left first naming convention takes precedence over the top first naming convention.

Figure 17c shows all of the virtual edge variables that are asserted true for the solution in Figure 15b; each virtual edge is annotated with the subnet that it is associated with. Virtual edges associated with subnets belonging to the same net are allowed to overlap;

this happens in two places. The two subnets belonging to Net 0 diverge from each other at grid location (0, 2). Each terminal is connected to exactly one virtual edge for a subnet or no virtual edges for that subnet. Non-terminals are connected to exactly 2 virtual edges from a subnet or no virtual edges from that subnet.

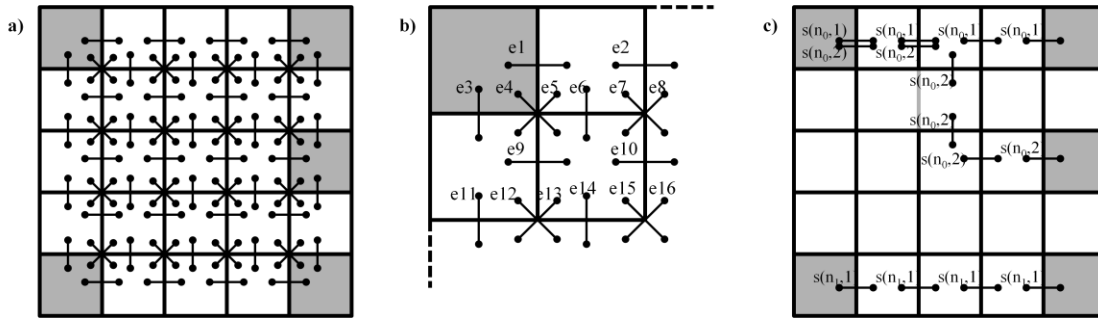


Figure 17: Virtual edges for the problem presented in Figure 15. (a) All adjacent grid cell locations where a set of virtual edge variables must be defined. (b) A blown up view of the upper left hand corner of the grid with each virtual edge location labeled; the virtual edge variable names for these locations can be found in Table 2. (c) The virtual edges shown are asserted true for the solution shown in Figure 15b.

Edge Location	Virtual Edge Variables
e4	ve_v0_0_0_to_1_1 ve_v1_0_0_to_1_1
e6	ve_v0_0_1_to_1_1 ve_v1_0_1_to_1_1
e8	ve_v0_1_1_to_0_2 ve_v1_1_1_to_0_2
e9	ve_v0_1_0_to_1_1 ve_v1_1_0_to_1_1
e10	ve_v0_1_1_to_1_2 ve_v1_1_1_to_1_2
e13	ve_v0_2_0_to_1_1 ve_v1_2_0_to_1_1
e14	ve_v0_1_1_to_2_1 ve_v1_1_1_to_2_1
e15	ve_v0_1_1_to_2_2 ve_v1_1_1_to_2_2

Table 2: Virtual edge naming conventions demonstrated for some of the virtual edge locations in Figure 17b.

5.1.2 Constraint 1: Terminal Valid Bit Constraint

Grid cells assigned to the zero terminal, t_0 , from any net are assigned to every one of that net's subnets. This is achieved by asserting the corresponding valid bits for those subnets at that grid location. The corresponding valid bits for any non-existing subnets are set to false. All other terminals are assigned only to their respective subnets; this is again achieved by appropriately asserting/de-asserting the respective valid bits.

5.1.2.1 Boolean Formulation for Constraint 1

Zero Terminal: $v_0 \cdot \dots \cdot v_{\hat{n}_i} \cdot \overline{v_{\hat{n}_i+1}} \cdot \dots \cdot \overline{v_{T-1}}$

Terminal i ($i > 0$): $\overline{v_0} \cdot \dots \cdot \overline{v_{i-1}} \cdot v_i \cdot \overline{v_{i+1}} \cdot \dots \cdot \overline{v_{T-1}}$

5.1.2.2 Remarks on Constraint 1

Each net's zero terminal will have a large impact on the final solution; this is because all subnets are arranged through their net's zero terminal. The empirical SAT implementation developed in Chapter 6 arbitrarily assigns the first terminal specified for every net as the zero terminal; therefore the user can control the zero terminals in their problem specification if they are aware of this.

5.1.2.3 Example of Constraint 1

Referring back to Figure 15, constraint 1 is applied as follows (using the nomenclature from Section 5.1.1):

1. At Grid location (0, 0):
 - a. $v_{0_0_0}$ is set to true
 - b. $v_{1_0_0}$ is set to true

2. At Grid location (0, 4):
 - a. v0_0_4 is set to true
 - b. v1_0_4 is set to false
3. At Grid location (2, 4):
 - a. v0_2_4 is set to false
 - b. v1_2_4 is set to true
4. At Grid location (4, 0):
 - a. v0_4_0 is set to true
 - b. v1_4_0 is set to false
5. At Grid location (4, 4):
 - a. v0_4_4 is set to true
 - b. v1_4_4 is set to false

5.1.3 Constraint 2: Terminal Arrangement Constraint

For each valid bit asserted at a specific terminal location, exactly one of that bit's respective virtual edge variables is also asserted. For any non-asserted valid bits at that grid location, all of that valid bit's respective edge variables are set to false.

5.1.3.1 Boolean Formulation for Constraint 2

Legal valid bits:

$$\begin{aligned}
 & \overline{v_{x,y}} \rightarrow \\
 & (\overline{ve_{v_{xm1_ym1_to_x,y}}} \cdot \overline{ve_{v_{xm1_y_to_x,y}}} \dots \overline{ve_{v_{x,y_to_xp1_y}}} \cdot \overline{ve_{v_{x,y_to_xp1_yp1}}} + \\
 & \overline{ve_{v_{xm1_ym1_to_x,y}}} \cdot \overline{ve_{v_{xm1_y_to_x,y}}} \dots \overline{ve_{v_{x,y_to_xp1_y}}} \cdot \overline{ve_{v_{x,y_to_xp1_yp1}}} + \\
 & \dots \\
 & \overline{ve_{v_{xm1_ym1_to_x,y}}} \cdot \overline{ve_{v_{xm1_y_to_x,y}}} \dots \overline{ve_{v_{x,y_to_xp1_y}}} \cdot \overline{ve_{v_{x,y_to_xp1_yp1}}} + \\
 & \overline{ve_{v_{xm1_ym1_to_x,y}}} \cdot \overline{ve_{v_{xm1_y_to_x,y}}} \dots \overline{ve_{v_{x,y_to_xp1_y}}} \cdot \overline{ve_{v_{x,y_to_xp1_yp1}}})
 \end{aligned}$$

Illegal valid bits:

$$\begin{aligned}
 & \overline{v_{x,y}} \rightarrow \\
 & (\overline{ve_{v_{xm1_ym1_to_x,y}}} \cdot \overline{ve_{v_{xm1_y_to_x,y}}} \dots \overline{ve_{v_{x,y_to_xp1_y}}} \cdot \overline{ve_{v_{x,y_to_xp1_yp1}}})
 \end{aligned}$$

For the formulation:

- a. $xm1$ refers to the terminal's x-coordinate minus 1; $xp1$ refers to the terminal's x-coordinate plus 1. Similarly $ym1$ and $yp1$ can be translated to $y-1$ and $y+1$ respectively (this convention is used throughout the rest of the paper).
- b. This formula must be enumerated for every valid bit.

The variable $v_{x,y}$ represents a specific valid bit at grid location (x, y) . A grid location has at most 8 adjacent grid cells. The connectivity of an adjacent grid cell through the subnet associated with valid bit v is described by a specific virtual edge variable (according to the naming conventions defined in section 5.1.1.2).

If $v_{x,y}$ is asserted true then exactly one virtual edge associated with valid bit v and attached to grid location (x, y) must be asserted true; this sentence is formulated by creating up to 8 conjunctions all *OR*'d together. Each conjunction joins all of the virtual edge variables together with all but one of them negated. In the second line of the "Legal valid bit" formulation the variable $ve_{v,xm1,y_to_x,y}$ indicates that the subnet associated with valid bit v will be assigned to both grid location (x, y) and grid location $(x-1, y)$; this subnet will be assigned to no other grid locations adjacent to (x, y) .

When $v_{x,y}$ is false all virtual edge variables associated with valid bit v and attached to grid location (x, y) are forced to false (as shown in the "Illegal valid bits" formulation).

5.1.3.2 Remarks on Constraint 2

The Boolean representation of this constraint will be smaller for terminals defined along the border or in the corners of the grid; this is because border and corner grid cell locations have fewer neighbors to connect to.

5.1.3.3 Example of Constraint 2

Referring back to Figure 15, constraint 2 is applied as shown below (using the nomenclature from Section 5.1.1). Grid location $(0, 0)$ has three adjacent grid locations meaning that the “Legal valid bit” formulation is defined by 3 conjunctions of 3 virtual edge variables *OR*'d together. The variable $ve_v0_0_0_to_0_1$ indicates that $s(n_0, 1)$ is assigned to the grid cell located adjacent and to the right of terminal $(0, 0)$. Similarly the variable $ve_v0_0_0_to_1_0$ indicates that $s(n_0, 1)$ is assigned to the grid cell located adjacent and below terminal $(0, 0)$. Finally the variable $ve_v0_0_0_to_1_1$ indicates that $s(n_0, 1)$ is assigned to the grid cell located adjacent, below, and to the right of terminal $(0, 0)$. An example of the “Illegal valid bit” formulation is formulated for grid location $(0, 4)$; no virtual edge variable associated with subnet $s(n_0, 2)$ can ever be true at this location.

1. At Grid location $(0, 0)$:
 - a. For valid bit 0 (which is true):

$$\overline{ve_v0_0_0_to_0_1} \cdot \overline{ve_v0_0_0_to_1_0} \cdot \overline{ve_v0_0_0_to_1_1} +$$

$$\overline{ve_v0_0_0_to_0_1} \cdot \overline{ve_v0_0_0_to_1_0} \cdot \overline{ve_v0_0_0_to_1_1} +$$

$$\overline{ve_v0_0_0_to_0_1} \cdot \overline{ve_v0_0_0_to_1_0} \cdot \overline{ve_v0_0_0_to_1_1}$$
 - b. For valid bit 1 (which is true):

$$\overline{ve_v1_0_0_to_0_1} \cdot \overline{ve_v1_0_0_to_1_0} \cdot \overline{ve_v1_0_0_to_1_1} +$$

$$\overline{ve_v1_0_0_to_0_1} \cdot \overline{ve_v1_0_0_to_1_0} \cdot \overline{ve_v1_0_0_to_1_1} +$$

$$\overline{ve_v1_0_0_to_0_1} \cdot \overline{ve_v1_0_0_to_1_0} \cdot \overline{ve_v1_0_0_to_1_1}$$
2. At Grid location $(0, 4)$:
 - a. For valid bit 0 (which is true):

$$ve_v0_0_3_to_0_4 \cdot \overline{ve_v0_1_3_to_0_4} \cdot \overline{ve_v0_0_4_to_1_4} +$$

$$\frac{\overline{\text{ve_v0_0_3_to_0_4}} \cdot \overline{\text{ve_v0_1_3_to_0_4}} \cdot \overline{\text{ve_v0_0_4_to_1_4}} + \overline{\text{ve_v0_0_3_to_0_4}} \cdot \overline{\text{ve_v0_1_3_to_0_4}} \cdot \overline{\text{ve_v0_0_4_to_1_4}}}{}$$

- b. For valid bit 1 (which is false):

$$\overline{\text{ve_v1_0_3_to_0_4}} \cdot \overline{\text{ve_v1_1_3_to_0_4}} \cdot \overline{\text{ve_v1_0_4_to_1_4}}$$

3. At Grid location (2, 4):

- a. For valid bit 0 (which is false):

$$\overline{\text{ve_v0_1_3_to_2_4}} \cdot \overline{\text{ve_v0_1_4_to_2_4}} \cdot \overline{\text{ve_v0_2_3_to_2_4}} \cdot \overline{\text{ve_v0_3_3_to_2_4}} \cdot \overline{\text{ve_v0_2_4_to_3_4}}$$

- b. For valid bit 1 (which is true):

$$\begin{aligned} & \overline{\text{ve_v1_1_3_to_2_4}} \cdot \overline{\text{ve_v1_1_4_to_2_4}} \cdot \overline{\text{ve_v1_2_3_to_2_4}} \cdot \overline{\text{ve_v1_3_3_to_2_4}} \cdot \overline{\text{ve_v1_2_4_to_3_4}} + \\ & \overline{\text{ve_v1_1_3_to_2_4}} \cdot \overline{\text{ve_v1_1_4_to_2_4}} \cdot \overline{\text{ve_v1_2_3_to_2_4}} \cdot \overline{\text{ve_v1_3_3_to_2_4}} \cdot \overline{\text{ve_v1_2_4_to_3_4}} + \\ & \overline{\text{ve_v1_1_3_to_2_4}} \cdot \overline{\text{ve_v1_1_4_to_2_4}} \cdot \overline{\text{ve_v1_2_3_to_2_4}} \cdot \overline{\text{ve_v1_3_3_to_2_4}} \cdot \overline{\text{ve_v1_2_4_to_3_4}} + \\ & \overline{\text{ve_v1_1_3_to_2_4}} \cdot \overline{\text{ve_v1_1_4_to_2_4}} \cdot \overline{\text{ve_v1_2_3_to_2_4}} \cdot \overline{\text{ve_v1_3_3_to_2_4}} \cdot \overline{\text{ve_v1_2_4_to_3_4}} \end{aligned}$$

4. At Grid location (4, 0):

- a. For valid bit 0 (which is true):

$$\begin{aligned} & \overline{\text{ve_v0_3_0_to_4_0}} \cdot \overline{\text{ve_v0_4_0_to_3_1}} \cdot \overline{\text{ve_v0_4_0_to_4_1}} + \\ & \overline{\text{ve_v0_3_0_to_4_0}} \cdot \overline{\text{ve_v0_4_0_to_3_1}} \cdot \overline{\text{ve_v0_4_0_to_4_1}} + \\ & \overline{\text{ve_v0_3_0_to_4_0}} \cdot \overline{\text{ve_v0_4_0_to_3_1}} \cdot \overline{\text{ve_v0_4_0_to_4_1}} \end{aligned}$$

- b. For valid bit 1 (which is false):

$$\overline{\text{ve_v1_3_0_to_4_0}} \cdot \overline{\text{ve_v1_4_0_to_3_1}} \cdot \overline{\text{ve_v1_4_0_to_4_1}}$$

5. At Grid location (4, 4):

- a. For valid bit 0 (which is true):

$$\begin{aligned} & \overline{\text{ve_v0_3_3_to_4_4}} \cdot \overline{\text{ve_v0_3_4_to_4_4}} \cdot \overline{\text{ve_v0_4_3_to_4_4}} + \\ & \overline{\text{ve_v0_3_3_to_4_4}} \cdot \overline{\text{ve_v0_3_4_to_4_4}} \cdot \overline{\text{ve_v0_4_3_to_4_4}} + \\ & \overline{\text{ve_v0_3_3_to_4_4}} \cdot \overline{\text{ve_v0_3_4_to_4_4}} \cdot \overline{\text{ve_v0_4_3_to_4_4}} \end{aligned}$$

- b. For valid bit 1 (which is false):

$$\overline{\text{ve_v1_3_3_to_4_4}} \cdot \overline{\text{ve_v1_3_4_to_4_4}} \cdot \overline{\text{ve_v1_4_3_to_4_4}}$$

5.1.4 Constraint 3: Non-Terminal Arrangement Constraint

If a valid bit assigned to a non-terminal cell is asserted true then exactly two of that valid bit's attached virtual edge variables are asserted true; these two virtual edge variables cannot be vertically or horizontally adjacent to each other. If a valid bit assigned to a non-terminal cell is asserted false then all of that valid bit's respective virtual edge variables are also false.

5.1.4.1 Boolean Formulation for Constraint 3

The Boolean formulation for this constraint is very similar to Constraint 2's. Instead of simply comprising up to 8 *OR* operations it consist of up to 20 *OR* operations; for this reason it is not shown here.

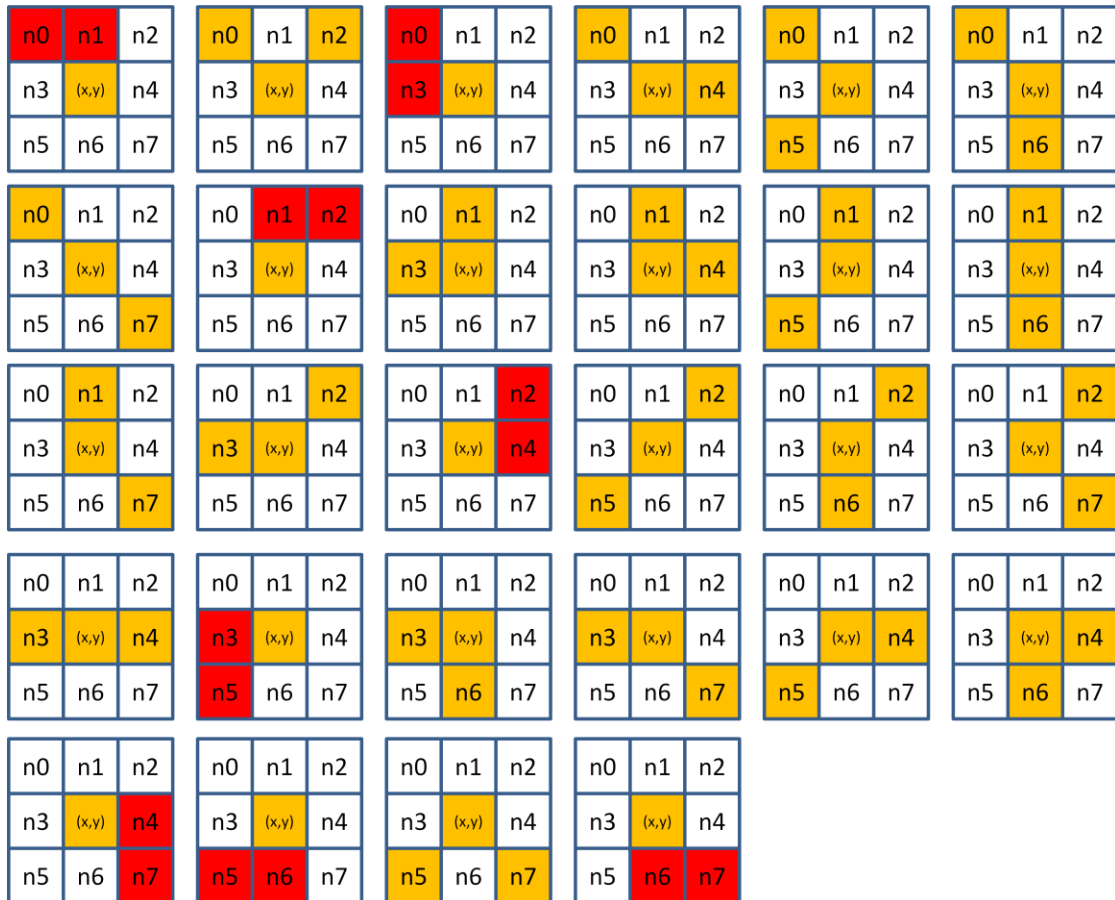


Figure 18: Valid subnet connections for non-terminal QCA cells. The configurations with cells highlighted in red are invalid; all other configurations are valid.

5.1.4.2 Remarks on Constraint 3

28 different sub-terminal arrangement configurations are shown in Figure 18; of these 8 are illegal. These configurations are illegal to avoid false connections. As described in

5.1.5 Theorem 1: Terminal Connectivity

By satisfying constraints 1-3, every terminal on the QCA grid is connected to at least one other terminal.

5.1.5.1 Proof 1: Terminal Connectivity

Constraint 1 restricts the connectivity of all terminals to one virtual edge and Constraint 2 restricts the connectivity of all non-terminals to exactly two virtual edges. Constraints 1 and 2 dictate that a terminal cannot be a part of a loop [8], therefore all terminals must be connected to other terminals for Constraints 1-3 to be satisfied [8].

5.1.6 Adjacent Cell Placement Problem

Constraints 1-3 are based on the formulation developed in [8]. For the traditional electronics frameworks, like the one presented in [8], adjacent grid points are connected by physical edges. For the QCA cell arrangement problem adjacent grid cells are connected by definition; edges are an abstraction created to describe the arrangement problem and have no physical representation.

Figure 19 illustrates an undesirable QCA layout that satisfies Constraints 1-3. The problem occurs because adjacent valid bits can be asserted without their corresponding virtual edge being asserted to indicate that a connection has been made. When valid bits at grid locations $(0, 1)$ and $(1, 0)$ are asserted the virtual edge variables should indicate that both of these grid locations are connected to $(0, 0)$; as illustrated in Figure 19 this is

not necessarily the case. An additional constraint restricting the connectivity of adjacent cells is created to combat this problem.

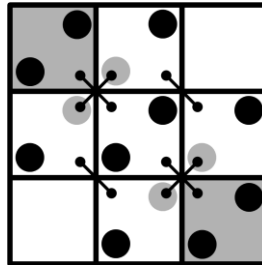


Figure 19: An undesirable arrangement produced by Constraints 1-3; the virtual edges that result in this placement are shown. The reader can verify that none of these constraints have been violated.

5.1.7 Constraint 4: Adjacent Cell Restriction Constraint

If two QCA cells belonging to the same subnet are placed adjacent to each other then exactly one of the following conditions must be true:

1. The two QCA cells must be connected directly via a virtual edge.
2. The two QCA cells must be connected to each other through exactly one other QCA cell via virtual edges.

5.1.7.1 Boolean Formulation for Constraint 4

A grid location has up to 8 adjacent cells (upper left, upper, upper right, left, right, lower left, lower, lower right). Because of symmetry only 4 of the possible 8 adjacent cell placements need to be restricted if this constraint is applied to all grid cells.

The formulation for the diagonally adjacent grid cells requires some explanation; it can be explained by examining the “Upper Right” formulation in more detail. The grid cell

located diagonally to the upper right of grid location (x, y) is located at $(x-1, y+1)$. This cell can be connected directly to grid location (x, y) in which case the virtual edge variable $ve_v_x_y_xm1_yp1$ must be true. This cell can also be connected indirectly through the grid location located adjacent and to the right of grid location (x, y) ; in this case the vertical and horizontal portions of the formulation for constraint 4 will enforce required virtual edge connections. The third case where the “Upper Right” grid cell is connected to grid location (x, y) through the “Upper” grid cell is analogous to the second situation just described. The *XOR* operator implements the “exactly one other” wording in condition 2 of Constraint 4 by forcing the placement of v_xm1_y and v_x_yp1 to be mutually exclusive of each other.

Upper Right:

$$v_xm1_yp1 \rightarrow (ve_v_x_y_to_xm1_yp1 + (v_xm1_y \oplus v_x_yp1))$$

Right:

$$v_x_yp1 \rightarrow ve_v_x_y_to_x_yp1$$

Lower:

$$v_x_yp1 \rightarrow ve_v_x_y_to_xp1_y$$

Lower Right:

$$v_xp1_yp1 \rightarrow (ve_v_x_y_to_xp1_yp1 + (v_xp1_y \oplus v_x_yp1))$$

5.1.7.2 Remarks on Constraint 4

With this constraint in place adjacent QCA cells only exist with their respective virtual edges as described in Section 2.2; this means that Constraints 1-3 will guarantee arrangements that fit the interconnect paradigms defined in this paper.

5.1.7.3 Example of Constraint 4

This constraint is applied to all 25 grid cells in Figure 15. It is only illustrated for one grid location (0, 1). The upper right constraint does not apply because (0, 1) is on the top border of the grid and no cells can be placed above it.

Adjacent Right:

$$v0_0_2 \rightarrow ve_v0_0_1_to_0_2$$

$$v1_0_2 \rightarrow ve_v1_0_1_to_0_2$$

Adjacent Lower:

$$v0_1_1 \rightarrow ve_v0_0_1_to_1_1$$

$$v1_1_1 \rightarrow ve_v1_0_1_to_1_1$$

Adjacent Lower Right:

$$v0_1_2 \rightarrow (ve_v0_0_1_to_1_2 + (v0_1_1 \oplus v0_0_2))$$

$$v1_1_2 \rightarrow (ve_v1_0_1_to_1_2 + (v1_1_1 \oplus v1_0_2))$$

5.1.8 Relationship between Terminals and their Respective Nets

Constraints 1-4 provide the infrastructure to achieve interconnects between terminals; they do nothing to differentiate which terminals are allowed to be connected together. Figure 20 demonstrates how Constraints 1-4 allow any combination of terminals to be connected together regardless of what net they belong to; this is because thus far Constraints 1-4 have lacked any constructs to differentiate between nets. This problem can be solved by assigning each terminal a net ID according to what net that terminal belongs to; a rule (constraint) is then put in place that only allows grid cells assigned to the same net ID to be placed contiguously. This technique was originally developed for the traditional electronics framework in [8].

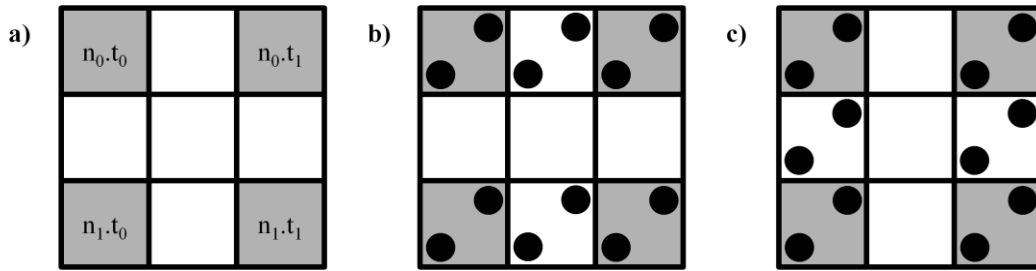


Figure 20: Given the specifications in (a) both (b) and (c) satisfy constraints 1-4. An additional constraint must be added to limit the connectivity between terminals to those within the same net.

5.1.8.1 The Net ID Vector and its Boolean Representation

For each grid cell, c , on the QCA grid, a variable vector $\vec{I}(c)$ is defined to identify any net crossing over that cell; the length of $\vec{I}(c)$ is a function of the number of nets, N , specified.

$$\vec{I}(c) = I_0(c)I_1(c)\dots I_{w-1}(c), \quad \text{where } w = \log_2(N)$$

The Net ID vector, $\vec{I}(c)$, serves two important purposes:

1. It ensures that distinct Nets are not shorted together.
2. Assigning terminals that belong to the same net the same Net ID ensures that all terminals belonging to a net will be connected to each other and not to terminals assigned a different Net ID.

5.1.9 Constraint 5: Terminal Net ID Constraint

$\vec{I}(c)$ is encoded at each terminal location so that its binary value matches that terminal's net ID.

5.1.9.1 Remarks on Constraint 5

All terminals that belong to the same net will have identical encodings for their ID vectors, $\vec{I}(c)$. The ID vector grows logarithmically in length as more nets are specified.

5.1.9.2 Example of Constraint 5

For the problem specified in Figure 15 there are two nets so the ID vector is only 1 bit

($w=1$). Constraint 5 is applied as follows:

1. At Grid location (0, 0):
ID0_0_0 is set to false
2. At Grid location (0, 4):
ID0_0_4 is set to false
3. At Grid location (2, 4):
ID0_2_4 is set to false
4. At Grid location (4, 0):
ID0_4_0 is set to true
5. At Grid location (4, 4):
ID0_4_4 is set to true

5.1.10 Constraint 6: Non-Terminal Net ID Constraint

Adjacent placement of two QCA cells implies that they will both have identical net ID

vectors, $\vec{I}(c)$.

5.1.10.1 Boolean Formulation for Constraint 6

In the formulation below, the ID variable represents the bit vector, $\vec{I}(c)$; ID variables are made identical with the if-and-only-if (*IFF*) operator. The *IFF* relation between ID variables is expanded into vector form by applying the *IFF* operator to each pair of corresponding ID bits and combining all bit relations via conjunction:

$$ID_{x,y} \leftrightarrow ID_{xm1_yp1} = (ID_{x,y}(0) \leftrightarrow ID_{xm1_yp1}(0)) \cdot \dots \cdot (ID_{x,y}(w-1) \leftrightarrow ID_{xm1_yp1}(w-1))$$

Any variable taking the form val_{x_y} is a disjunction of all valid bits at grid location (x, y) : $val_{x_y} = v0_{x_y} + \dots + vTm1_{x_y}$. Because of symmetry this constraint needs to be encoded for 4 of the possible 8 adjacent cell placements if this constraint is applied to all grid cells.

Upper Right:

$$(val_{x_y} \cdot val_{xm1_yp1}) \rightarrow (ID_{x_y} \leftrightarrow ID_{xm1_yp1})$$

Right:

$$(val_{x_y} \cdot val_{x_yp1}) \rightarrow (ID_{x_y} \leftrightarrow ID_{x_yp1})$$

Lower:

$$(val_{x_y} \cdot val_{xp1_y}) \rightarrow (ID_{x_y} \leftrightarrow ID_{xp1_y})$$

Lower Right:

$$(val_{x_y} \cdot val_{xp1_yp1}) \rightarrow (ID_{x_y} \leftrightarrow ID_{xp1_yp1})$$

5.1.10.2 Remarks on Constraint 6

The size of the ID vector grows logarithmically with the number of nets specified.

5.1.10.3 Example of Constraint 6

For the problem specified in Figure 15 this constraint is illustrated for one grid location, $(0, 1)$. The upper right constraint does not apply because $(0, 1)$ is on the top border of the grid and no cells can be placed above it. The ID vector for this example is only 1 bit because there are only 2 nets. In this example val_{0_1} is equal to $v0_{0_1} + v1_{0_1}$.

Right:

$$(val_{0_1} \cdot val_{0_2}) \rightarrow (ID0_{0_1} \leftrightarrow ID0_{0_2})$$

Lower:

$$(val_{0_1} \cdot val_{1_1}) \rightarrow (ID0_{0_1} \leftrightarrow ID0_{1_1})$$

Lower Right:

$$(val_{0_1} \cdot val_{1_2}) \rightarrow (ID0_{0_1} \leftrightarrow ID0_{1_2})$$

5.2 Constraints to Prevent Subnet Interference

A solution satisfying constraints 1-6 will successfully arrange all subnets that make up the specified multi-terminal nets; this will be achieved without contact between any of the nets. Placing all of the subnets on the same grid may result in ambiguous arrangements; this is likely because no constraints have been defined to prevent unwanted interference between QCA subnets. Potential subnet interference problems are examined and additional constraints are formulated to avoid them.

5.2.1 Adjacent Sub-Nets

Two sub-nets are adjacent when they are assigned to adjacent grid locations after they have diverged from each other. This situation is illustrated in Figure 21b; the two subnets split off from each other at grid location (0, 1) and are adjacent to each other by the following two pairs of grid locations:

1. (1,1); (0,2)
2. (1,2); (0,2)

Though this configuration may successfully connect all of the terminals together it is undesirable because it does not fit any of the pre-defined interconnect paradigms from Section 2.2. This situation is avoided by adding another constraint.

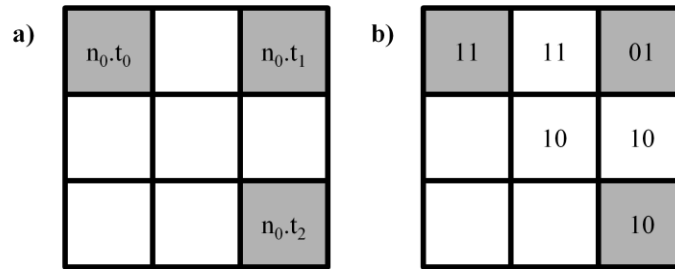


Figure 21: When sub-nets are allowed to touch undesirable cell arrangements can occur. (a) A single net consisting of 2 subnets is specified. (b) In this solution the 2 subnets diverge at grid location (0, 1). The resulting QCA cell placements do not fit any pre-determined interconnect paradigms. The 2 bit binary vectors are the valid bits; the right most bit is the valid bit for subnet 0.

5.2.2 Constraint 7: Adjacent Subnet Restriction

If a QCA cell, $c1$, is valid for a subnet, A , and not valid for another subnet, B , then it may be placed adjacent to a QCA cell, $c2$, that is valid for subnet B and not valid for subnet A only if $c1$ and $c2$ are both adjacent to a third QCA cell, $c3$, that is valid for both subnet A and subnet B . This logic is enumerated for all possible combinations of subnets for all non-terminal grid cells.

5.2.2.1 Boolean Formulation for Constraint 7

Adjacent subnets can be detected by examining the valid bits of two adjacent grid locations. The Boolean expression to detect two adjacent subnets, Net 0 and Net 1, is shown below; it consists of three *XOR* expressions to check three conditions:

1. One grid cell has valid bit 0 asserted and the other does not.
2. One grid cell has valid bit 1 asserted and the other does not.
3. One grid cell has either valid bit 0 or valid bit 1 asserted but not both (this condition along with condition 1 and 2 imply that the other grid cell will also have either valid bit 0 or valid bit 1 asserted but not both).

$$(v0_{x,y} \oplus v0_{\text{adjacent}}) \cdot (v1_{x,y} \oplus v1_{\text{adjacent}}) \cdot (v0_{x,y} \oplus v1_{\text{adjacent}})$$

For horizontally and vertically adjacent grid cells this condition is not allowed.

Right:

$$\overline{(v0_x_y \oplus v0_x_yp1) \cdot (v1_x_y \oplus v1_x_yp1) \cdot (v0_x_y \oplus v1_x_yp1)}$$

Lower:

$$\overline{(v0_x_y \oplus v0_xp1_y) \cdot (v1_x_y \oplus v1_xp1_y) \cdot (v0_x_y \oplus v1_xp1_y)}$$

For diagonally adjacent grid cells this condition is permitted if they are both connected by a grid cell that has both valid bit 0 and valid bit 1 asserted true; this can be translated into a Boolean expression with the implication operator as shown below. Adjacent grid cells assigned to different subnets imply that they are both adjacent to another grid cell assigned to both of these subnets.

Upper Right:

$$((v0_x_y \oplus v0_xm1_yp1) \cdot (v1_x_y \oplus v1_xm1_yp1) \cdot (v0_x_y \oplus v1_xm1_yp1)) \rightarrow ((v0_xm1_y \cdot v1_xm1_y) \oplus (v0_x_yp1 \cdot v1_x_yp1))$$

Lower Right:

$$((v0_x_y \oplus v0_xp1_yp1) \cdot (v1_x_y \oplus v1_xp1_yp1) \cdot (v0_x_y \oplus v1_xp1_yp1)) \rightarrow ((v0_xp1_y \cdot v1_xp1_y) \oplus (v0_x_yp1 \cdot v1_x_yp1))$$

5.2.2.2 Remarks on Constraint 7

This constraint is written so that two subnets may be assigned to the same cell (for the case where they have not yet diverged). Figure 22 illustrates QCA cell arrangements effected by this constraint. Subnets can only be adjacent to each other if they are both adjacent to a grid cell where they are both valid (a divergence point). In all other cases adjacent cells assigned to different subnets are not allowed.

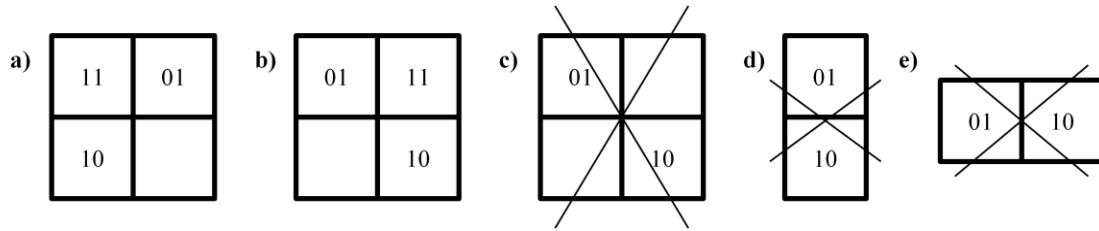


Figure 22: Adjacent subnet arrangements that are restricted by constraint 7. (a) and (b) are valid placements where subnets are diverging from each other. (c), (d), and (e) are all invalid QCA cell placements that result in adjacent subnets.

Constraint 7 is very expensive because it grows exponentially with respect to the maximum number of subnets T . Table 3 lists the combinations of valid bits for some different values of T .

Maximum # of subnets (T)	Adjacent valid bit combinations	# of enumerations	# of enumerations
1	-	-	-
2	(v0,v1)	$\frac{2 * 1}{2}$	2
3	(v0,v1); (v0,v2); (v1,v2)	$\frac{3 * 2}{2}$	3
4	(v0,v1); (v0,v2); (v0,v3); (v1,v2); (v1,v3); (v2,v3)	$\frac{4 * 3}{2}$	6
5	(v0,v1); (v0,v2); (v0,v3); (v0,v4); (v1,v2); (v1,v3); (v1,v4); (v2,v3); (v2,v4); (v3,v4)	$\frac{5 * 4}{2}$	10

Table 3: This table shows the combinations of valid bits that must be accounted for when formulating Constraint 7 which will grow exponentially with respect to the maximum number of subnets T .

5.2.2.3 Example of Constraint 7

This constraint is applied to every grid location. Its application is shown at grid location $(1, 1)$ for the problem presented in Figure 15. The constraint is enumerated over two combinations of subnets ($T=2$).

Upper Right:

$$(((v0_1_1 \oplus v0_0_2) \cdot (v1_1_1 \oplus v1_0_2) \cdot (v0_1_1 \oplus v1_1_1)) \rightarrow ((v0_0_1 \cdot v0_1_y) \oplus (v0_1_2 \cdot v1_1_2))) \cdot$$

Right:

$$((v0_1_1 \oplus v0_1_2) \cdot (v1_1_1 \oplus v1_1_2) \cdot (v0_1_1 \oplus v1_1_1)) \cdot$$

Lower:

$$((v0_1_1 \oplus v0_2_1) \cdot (v1_1_1 \oplus v1_2_1) \cdot (v0_1_1 \oplus v1_1_1)) \cdot$$

Lower Right:

$$(((v0_1_1 \oplus v0_2_2) \cdot (v1_1_1 \oplus v1_2_2) \cdot (v0_1_1 \oplus v1_1_1)) \rightarrow ((v0_2_1 \cdot v2_1_y) \oplus (v0_1_2 \cdot v1_1_2)))$$

5.2.3 Multi-Terminal Re-Convergence

Two subnets re-converge (join together) after they have diverged (separated) if they are both assigned to two or more non-adjacent cells; this leads to undesirable net sub-loops.

A constraint is added to prevent subnet re-convergence. This additional constraint also prevents a very similar situation where illegal valid bits inadvertently turn on and lead to net sub-loops. Examples of these two types of loops are shown in Figure 23.

11	11	11	01	11	11	11	01	01
		10		10		10		
		10	10	10		10	10	10
01	01	01	11	11	11	01	01	01
			10		10			
			10	10	10			

Figure 23: In the top net the two subnets diverge and then re-converge creating a loop within the net. In the bottom net an illegal valid bit flips on and creates a net loop similar to the re-convergence loop on the top.

5.2.4 Constraint 8: Single Split Constraint

Subnets may split from their parent nets exactly one time. This is achieved by creating Boolean variables to represent a global split coordinate on the grid for every subnet. A split can only occur at a cell whose grid location matches the global split coordinate for that subnet.

5.2.4.1 Boolean Formulation for Constraint 8

Two subnets splitting from each other can be detected by checking for the following two conditions at every grid location:

1. A valid bit is asserted at the current grid location.
2. The same valid bit is not asserted at any adjacent grid location while another valid bit is asserted at that adjacent grid location.

In the Boolean formulation below $val_{x,y}$ indicates that any valid bit at (x, y) is asserted (it is the disjunction of all valid bits). The variable $s_{v,x,y}$ is the Boolean representation of condition 2.

$$s_{v,x,y} = \overline{(\overline{val_{xm1,ym1} \rightarrow v_{xm1,ym1}} + \overline{val_{xm1,y} \rightarrow v_{xm1,y}} + \overline{val_{xm1,yp1} \rightarrow v_{xm1,yp1}} + \overline{val_{x,ym1} \rightarrow v_{x,ym1}} + \overline{val_{x,yp1} \rightarrow v_{x,yp1}} + \overline{val_{xp1,ym1} \rightarrow v_{xp1,ym1}} + \overline{val_{xp1,y} \rightarrow v_{xp1,y}} + \overline{val_{xp1,yp1} \rightarrow v_{xp1,yp1}})}$$

The split condition is only allowed to happen if the global split variable matches the x and y-coordinates of the split condition. This rule is enforced with the *IFF* operator. The phrase “ $split_ID == var$ ” is expanded so the $split_ID$ vector encodes the value of var .

$$(v_{x,y} \cdot s_{v,x,y}) \leftrightarrow ((split_ID_v_X == X) \cdot (split_ID_v_Y == Y))$$

5.2.4.2 Remarks on Constraint 8

Constraint 8 grows logarithmically with respect to the width of the grid, logarithmically with respect to the length of the grid, and linearly with respect to the total number of subnets specified. The length of the Split ID vector is determined by taking the base 2 logarithm of the number of grid rows/columns and rounding up.

5.2.4.3 Example of Constraint 8

This constraint is applied to every grid location. Its application is shown at grid location $(1, 4)$ for the problem presented in Figure 15. This constraint is enumerated twice ($T=2$).

The split ID's for the x-coordinate and y-coordinate are both 3 bits ($\lceil \log_2(5) \rceil$).

Enumeration 1:

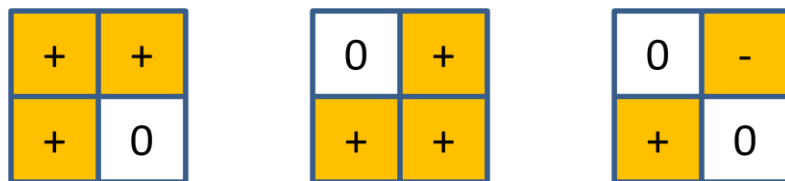
$$(v_{0,1,4} * s_{v,0,1,4}) \leftrightarrow \overline{(\overline{split_ID0_v0_X0} \cdot \overline{split_ID0_v0_X1} \cdot \overline{split_ID0_v0_X2} \cdot \overline{split_ID0_v0_Y0} \cdot \overline{split_ID0_v0_Y1} \cdot \overline{split_ID0_v0_Y2})}$$

Enumeration 2:

$$(v1_1_4 * s_v1_1_4) \leftrightarrow \frac{(\text{split_ID0_v1_X0} \cdot \text{split_ID0_v1_X1} \cdot \text{split_ID0_v1_X2} \cdot \text{split_ID0_v1_Y0} \cdot \text{split_ID0_v1_Y1} \cdot \text{split_ID0_v1_Y2})}{\text{split_ID0_v1_Y0} \cdot \text{split_ID0_v1_Y1} \cdot \text{split_ID0_v1_Y2}}$$

5.3 QCA Polarity

As discussed in Chapter 2, QCA cells have 2 different configurations or polarities. If two cells are located diagonal from each other without being connected by another adjacent cell then the polarity of the cells are opposite. In all other cases adjacent cells have the same polarity. For each grid cell a variable pol_{x_y} is created to represent that cell's QCA polarity. The polarity properties are illustrated in Figure 24. A constraint must be created to ensure that polarity rules are not violated and all terminals have the same polarity.



(a) Polarity transferred (b) Polarity transferred (c) Polarity inverted

Figure 24: The polarity of a QCA cell either transfers to its neighboring cells, or is inverted.

5.3.1 Constraint 9: Polarity Observance

All cells on the QCA grid transfer polarity as shown in Figure 24. All terminals belonging to the same net have identical polarities.

5.3.1.1 Boolean Formulation for Constraint 9

Part 1:

In the Boolean formulation below $val_{x,y}$ indicates if any valid bit at (x, y) is asserted (it is the disjunction of all valid bits). The *IFF* operator is used to transfer polarity and the *XOR* operator is used to switch polarity.

Upper Right:

$$(val_{xm1_yp1} \cdot \overline{val_{x_yp1}} \cdot \overline{val_{xm1_y}}) \rightarrow (pol_{x_y} \oplus pol_{xm1_yp1})$$

Right:

$$val_{x_yp1} \rightarrow (pol_{x_y} \leftrightarrow pol_{x_yp1})$$

Lower:

$$val_{xp1_y} \rightarrow (pol_{x_y} \leftrightarrow pol_{xp1_y})$$

Lower Right:

$$(val_{xp1_yp1} \cdot \overline{val_{x_yp1}} \cdot \overline{val_{xp1_y}}) \rightarrow (pol_{x_y} \oplus pol_{xp1_yp1})$$

An additional Boolean encoding is required to guarantee that all terminals belonging to the same net carry the same polarity. An example of this can be seen below in Section 5.3.1.3.

5.3.1.2 Remarks on Constraint 9

This constraint exists to support the inverting wire construct; it can be ignored if this structure is not used.

5.3.1.3 Example of Constraint 9

This constraint is applied to every grid location. Its application is shown at grid location $(1, 1)$ for the problem presented in Figure 15.

Upper Right:

$$(val_{0_2} \cdot \overline{val_{1_2}} \cdot \overline{val_{0_1}}) \rightarrow (pol_{1_1} \oplus pol_{0_2})$$

Right:

$$\text{val}_{1_2} \rightarrow (\text{pol}_{1_1} \leftrightarrow \text{pol}_{1_2})$$

Lower:

$$\text{val}_{2_1} \rightarrow (\text{pol}_{1_1} \leftrightarrow \text{pol}_{2_1})$$

Lower Right:

$$(\text{val}_{2_2} \cdot \overline{\text{val}_{1_2}} \cdot \overline{\text{val}_{2_1}}) \rightarrow (\text{pol}_{1_1} \oplus \text{pol}_{2_2})$$

In addition the terminals in Figure 15 must have the same polarities:

$$(\text{pol}_{0_0} \leftrightarrow \text{pol}_{0_4}) \cdot (\text{pol}_{0_0} \leftrightarrow \text{pol}_{2_4}) \cdot (\text{pol}_{4_0} \leftrightarrow \text{pol}_{4_4})$$

5.4 Minimization Constraint

Constraints 1-9 guarantee a solution to the 90 degree QCA cell arrangement problem but there are still some improvements that can be made. Two problems were observed while experimentally verifying Constraints 1-9:

1. Random Wire Loops
2. Inefficient Cell Arrangements

These problems and their origins are described in Sections 5.4.1 and 5.4.2. These problems can be solved by converting the Boolean formulation into a Pseudo-Boolean ILP formulation and adding a minimization constraint; this strategy is discussed in Section 5.4.3.

5.4.1 Random Wire Loops

Constraints 1-9 allow cell arrangements solutions with random wire loops. These wire loops are not connected to any other interconnects. This happens because each cell belonging to one of these wire loops is a non-terminal connected to its neighbors via two

virtual edges. Figure 25 shows an example layout with a random wire loop. Generally wire loops are undesirable; they are unintended structures with no benefits. These structures could feasibly be removed by some sort of post-processing tool but another solution is suggested in section 5.4.3.

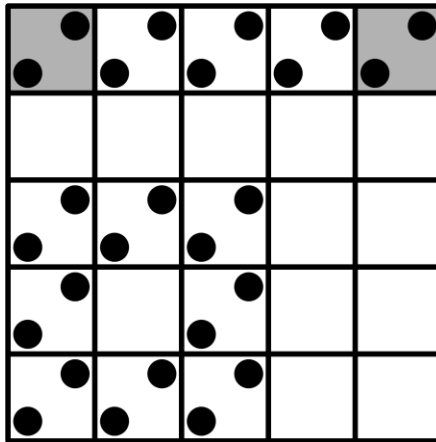


Figure 25: An example QCA placement that satisfies Constraints 1-9 and has a wire loop.

5.4.2 Inefficient Cell Arrangements

Constraints 1-9 do not differentiate between good solutions and bad solutions for the QCA cell arrangement problem; a SAT solver will return the first solution that it finds completely ignoring the quality of the solution. Figure 26 shows two valid cell arrangements for the same set of terminals. The solution in Figure 26a is probably easier to fabricate and more noise immune than the one in Figure 26b; a SAT solver is just as likely to return either solution given Constraints 1-9.

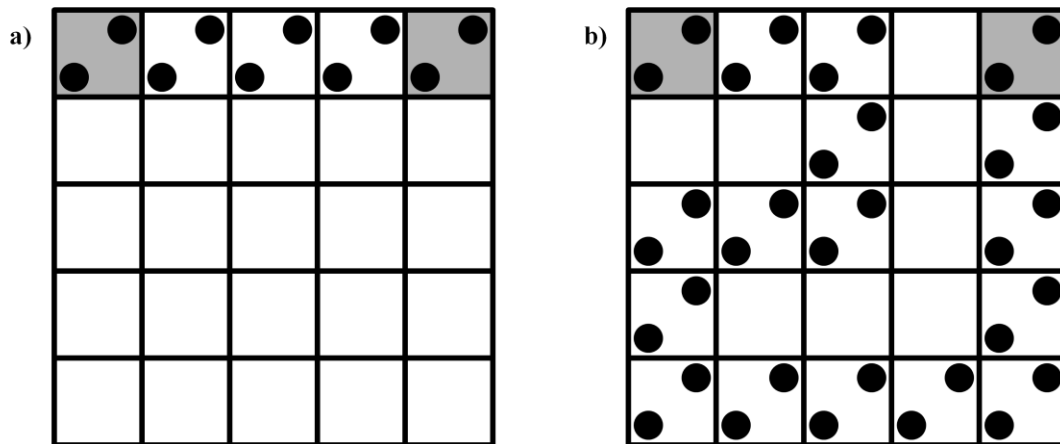


Figure 26: Two valid cell arrangements according to constraints 1-9. (a) The most efficient cell arrangement for the two specified terminals. (b) One of the most inefficient ways to form an interconnect for the two specified terminals.

5.4.3 Eliminating Random Wire Loops and Inefficient Cell Arrangements

Both random wire loops and inefficient cell arrangements can be eliminated by converting the CNF representation of Constraints 1-9 into their equivalent ILP representation, adding a minimization constraint, and using a Pseudo-Boolean ILP solver to obtain a solution to the 90 degree QCA cell arrangement problem.

5.4.3.1 Constraint 10

Minimize diagonally adjacent QCA cell placements.

5.4.3.2 Formulation for Constraint 10

In the Boolean formulation below $val_{x,y}$ indicates if any valid bit at (x, y) is asserted (it is the disjunction of all valid bits). Diagonally adjacent QCA cell placements are detected and their Tseitin output variables are accumulated; the sum of these output variables is then specified as the minimization constraint to the pseudo-Boolean solver.

Upper Right:

$$\text{con_upper_right_x_y} = (\text{val_x_y} \cdot \text{val_xm1_yp1})$$

Lower Right:

$$\text{con_lower_right_x_y} = (\text{val_x_y} \cdot \text{val_xp1_yp1})$$

$$\text{minimize}(\sum(\text{con_upper_right_x_y} + \text{con_lower_right_x_y}))$$

5.4.3.3 Remarks on Constraint 10

This constraint works by minimizing the number of turns in a QCA wire; a turn always occurs in the presence of two diagonally adjacent QCA cells. If the terminals are specified along the borders of the grid this constraint will always result in minimum length wires. If the terminals are not on the borders then the cell arrangements may not be minimum length; this can be mitigated by limiting the valid virtual edges that a terminal can attach to.

The implementation of constraint 10 eliminates the need to specify constraint 9. This is because in order for two subnets to diverge they must turn away from each other. If more than one divergence were to occur then constraint 10 would not be met; therefore constraint 9 is simply not needed in the presence of the minimization constraint.

5.5 Specifying Keep-Outs

Sometimes it is desirable to specify keep-outs. A keep-out is a portion on the grid where QCA cells are not allowed to be placed. This can easily be accomplished by forcing all valid bits associated with grid locations within the keep-out area to false.

Chapter 6 - Constraint Implementation

A Boolean logic formulation for the QCA cell arrangement problem is realized by translating constraints 1-9 into their equivalent Boolean representation; it can then be verified for correctness with a SAT solver or a Pseudo-Boolean ILP solver. This is not a trivial task - the Boolean formulation is effected by the grid dimensions, number of nets, and the largest number of terminals assigned to any net.

6.1 SAT Based Solution

The tool flow for generating a satisfiability formulation specific to the QCA cell arrangement problem and processing it is shown in Figure 27a. For any given problem, the Boolean formulation is first generated directly and then converted into CNF format. Two Perl scripts were written to achieve this task. The first Perl file, `qca_bool.pl`, generates the Boolean representation of constraints 1-9 for any specified problem. The second Perl file, `cnf_gen.pl`, converts the Boolean formula into CNF format by implementing the Boolean to CNF compiler described in section 6.3. The CNF representation is then passed to a SAT solver (MiniSat) which returns a solution to the problem.

In general `qca_bool.pl` applies constraints to the all of the relevant grid locations, implements constraint enumeration, and expands bit vectors. A grid and set of terminals are defined in a grid file. The grid file is read by `qca_bool.pl`. The original intent was to specify all constraints in the constraint file; this turned out to be not practical because some of the constraints were too complicated. Constraints 1-6 and 9 are specified in the

constraint file. Constraint 7 is generated by `qca_bool.pl` directly. Constraint 8 is partially specified in the constraint file and partially generated directly by `qca_bool.pl`. `qca_bool.pl` returns Constraints 1-9 in Boolean form (as a `bool` file) written specifically for the problem given in the grid file. The `bool` file is then passed to `cnf_gen.pl` which converts the Boolean specification into its equisatisfiable CNF representation (as a CNF file). The CNF file can then be processed by a SAT solver (MiniSat) which returns the final solution in a text file; this text file must be trivially parsed to get the final QCA placement that will guarantee a solution to the QCA cell arrangement problem.

6.2 Pseudo-Boolean Based Solution

A pseudo-Boolean solver is a SAT solver extended to solve linear constraints over Boolean variables [20]. It can also enforce an optimization constraint (allowing Constraint 10 to be implemented).

The tool flow for generating a pseudo-Boolean formulation specific to the QCA cell arrangement problem and processing it is shown in Figure 27b; the first two steps generate a CNF representation of constraints 1-9 (using the same methodology described in section 6.1). The CNF file is then passed into a Perl file called `opb_gen.pl`. `opb_gen.pl` first searches the comments in the CNF file for all Boolean variables indicating diagonally adjacent grid cells and adds them to the minimization constraint. It then reads every clause in the CNF file and converts them into an equivalent OPB format (as described in [18]). `opb_gen.pl` outputs an OPB file describing constraints 1-10 for the

problem specified in the grid file; this OPB file is passed into MiniSat+ which returns a solution to the QCA cell arrangement problem.

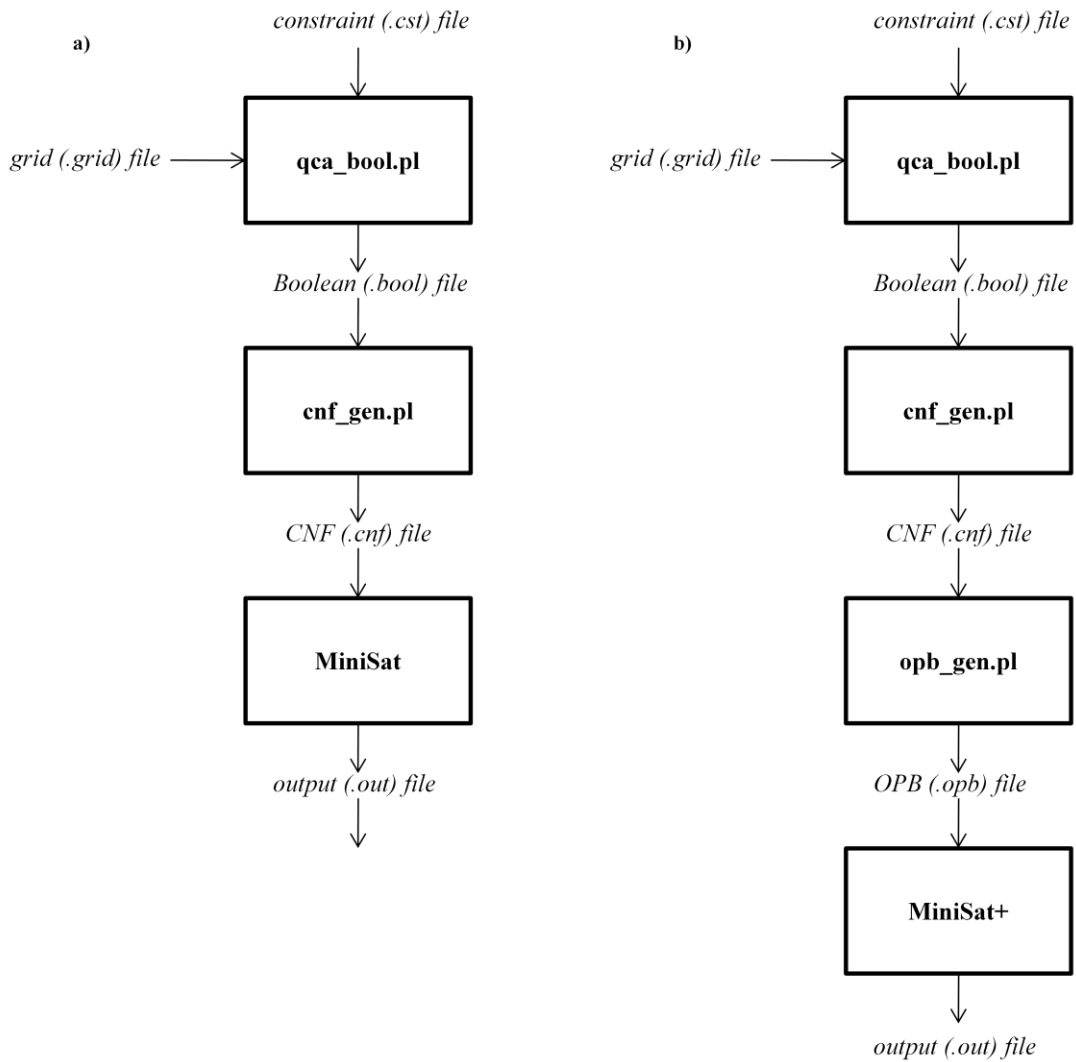


Figure 27: (a) Tool flow for generating and solving a SAT formulation. (b) Tool flow for generating and solving a Pseudo-Boolean ILP formulation.

6.3 Boolean to CNF Compiler

This section discusses the Boolean to CNF compiler that was written for this thesis. This tool is useful because most SAT solvers (like MiniSat) require their input Boolean formula to be specified in CNF format so they can take advantage of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. Boolean formulas can be converted to CNF's using Tseitin's transformation which converts an expression into a CNF by adding extra literals to the Boolean formula.

6.3.1 Tseitin's Transformation for Basic Operators

Tseitin's transformation works by treating basic logic operators like combinational gates; this is illustrated in Table 4. Each gate's output is related to its inputs via the if-and-only-if (IFF) operator. Any formula can be converted to a CNF by applying the CNF representations shown in Table 4 to each individual gate in a formula's equivalent combinational circuit; the CNF formulas for all gates are then *AND*'d together. Finally the last output of the equivalent combinational circuit is forced to true by adding a single literal clause; doing this means that any solution found by a SAT solver applies both to the new equation (with extra literals) and the original equation. This process is illustrated in Example 1.

It is worth noting that the CNF formula derived through Tseitin's transformation is different than the original formula – it contains many extra literals (Tseitin variables). CNF's obtained via Tseitin's methods are termed equisatisfiable because forcing the last output to true makes the formulas equivalent for the purposes of satisfiability. The values

obtained for the extra literals in the CNF formula can be ignored when applying the model returned by a SAT solver to the original formula.

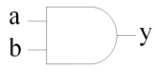


Operator	Symbols	Gate	Boolean Formula	Equivalent CNF
<i>and</i>	$*, \wedge$		$y \leftrightarrow a * b$	$(y + -a + -b)$ $(-y + a)$ $(-y + b)$
<i>or</i>	$+, \vee$		$y \leftrightarrow a + b$	$(-y + a + b)$ $(y + -a)$ $(y + -b)$
<i>not</i>	$-, \neg$		$y \leftrightarrow -a$	$(-y + -a)$ $(y + b)$

Table 4: Tseitin's transformation for the 3 fundamental Boolean operators. The *AND* and *OR* gates are shown with 2 inputs, but the formulas will scale trivially for any number of inputs.

6.3.1.1 Example 1

Convert the expression $\neg(A*B*C + D*E)$ into a CNF.

Figure 28 shows an equivalent logic circuit for the expression. Intermediate variables have been added to label nodes in the circuit that are not referenced in the equation; these intermediate variables will be used to form a CNF by applying the equations from Table 4.

<i>CNF for Y1:</i>	<i>CNF for Y2:</i>	<i>CNF for Y3</i>	<i>CNF for Y4</i>
$(y1 + -a + -b + -c)$	$(y2 + -d + -e)$	$(-y3 + y1 + y2)$	$(-y4 + -y3)$
$(-y1 + a)$	$(-y2 + d)$	$(y3 + -y1)$	$(y4 + y3)$
$(-y1 + b)$	$(-y2 + e)$	$(y3 + -y2)$	
$(-y1 + c)$			

All of the equivalent CNF's shown above are AND'd together. Finally Y4 is asserted true by adding an extra clause. The final equivalent CNF is:

$$\begin{aligned}
 &(y1 + -a + -b + -c) \\
 &(-y1 + a) \\
 &(-y1 + b) \\
 &(-y1 + c) \\
 &(y2 + -d + -e) \\
 &(-y2 + d) \\
 &(-y2 + e)
 \end{aligned}$$

$(-y3 + y1 + y2)$
 $(y3 + -y1)$
 $(y3 + -y2)$
 $(-y4 + -y3)$
 $(y4 + y3)$
 $(y4)$

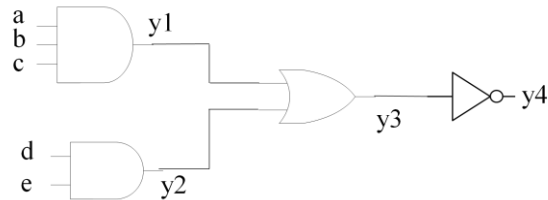


Figure 28: Equivalent combinational circuit for the formula $A*B*C + D*E$

6.3.2 Converting Higher Level Operations into CNF Format

The Boolean to CNF compiler discussed in this paper supports: *XOR*, implication, and *IFF*. These operators can be implemented by expanding them to equivalent formulas composed of basic operators, but it is more efficient to represent these operators directly as CNF expressions. Table 5 lists the optimum CNF expressions for these higher level operators.

Operator	Symbols	Gate	Boolean Formula	Equivalent CNF
<i>XOR</i>	$\oplus, \underline{\vee}, \wedge, @$		$y \leftrightarrow a \oplus b$	$(-y + a + b)$ $(-y + -a + -b)$ $(y + -a + b)$ $(y + a + -b)$
<i>implication</i>	\rightarrow		$y \leftrightarrow (a \rightarrow b)$	$(y + a + b)$ $(y + -a + -b)$ $(-y + -a + b)$ $(y + a + -b)$
<i>IFF</i>	\leftrightarrow		$y \leftrightarrow (a \leftrightarrow b)$	$(y + a + b)$ $(y + -a + -b)$ $(-y + -a + b)$ $(-y + a + -b)$

Table 5: CNF representations for XOR, implication, and IFF operations.

6.3.3 Implementation of the Boolean to CNF Compiler

The Boolean to CNF compiler performs the following algorithm:

1. Convert input Boolean Formula into its equivalent Parse Tree representation.
2. Traverse the Parse Tree and apply Tseitin's Transformation to every node.

6.3.4 The Structure of a Parse Tree

The Boolean to CNF compiler works by converting Boolean formulas into equivalent parse trees and evaluating them. Parse trees are acyclical graphs comprising nodes that represent either Boolean operations or literals. Figure 29 shows the parse tree for the expression $A*B*C + D*E$. The formula's literals are all at the bottom of the parse tree; these nodes are called leaf nodes. The internal structure of the parse tree is composed of operation nodes.

It is worth noting that the parse tree's structure is the same as the Boolean Formula's equivalent gate representation (Figure 28 without the last inverter). Each operation node can be mapped to a combinational gate. Each edge can be mapped to a gate's input or output. At the bottom most level of the parse tree, leaf nodes (literals) are connected to operator nodes. At intermediate levels operators nodes feed into other operator nodes. In reality all operation nodes have a literal associated with them - the Tseitin variable output by the operation node's equivalent gate and its inputs.

Table 6 and Table 7 describe the internal structures of a leaf node and an operation node respectively, as defined in the CNF to Boolean converter. The leaf node structure is very

simple; it contains a variable to store the name of its literal, an ID to reference it, and a pointer to its parent operation.

The operation node contains a string representing its Boolean operation, an ID to reference it, and a pointer to its parent operation. In addition, operation nodes must keep track of their children, which can be both leaf nodes and other operation nodes. The operation node structure maintains an array of children; children are defined by the Child structure. The Child structure is defined in Table 8; it consists of a type and an ID for the node that it represents. An operation node also maintains a counter to keep track of which children have been processed (converted into CNF form); this is useful in determining when the node itself is ready to be processed. Finally, the operation node contains a variable to hold its output Tseitin variable, which is determined when the node is processed.

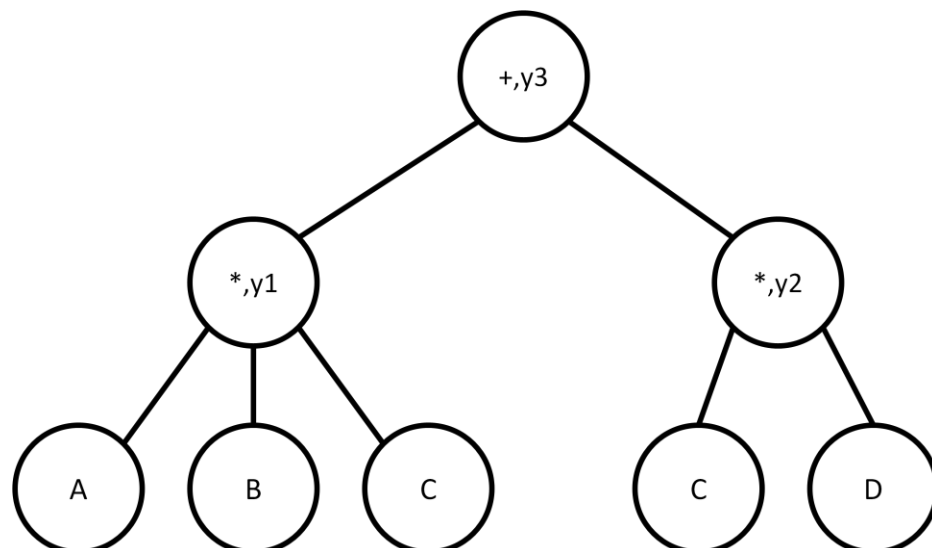


Figure 29: Parse tree for Boolean formula $A*B*C + D*E$. The operator nodes indicate both the operator and the equivalent gate's output Tseitin variable.

Leaf Node Structure	Description
String <i>var</i>	The name of the variable represented by this leaf node.
Integer <i>ID</i>	An ID used to reference this node.
Integer <i>parent_ID</i>	The ID of the operator that this node is associated with.

Table 6: The leaf node data structure contains 3 members.

Operation Node Structure	Description
String <i>op</i>	The operation implemented by this node. This will be a gate from Table 3 or Table 4.
Integer <i>ID</i>	An ID used to reference this node.
Integer <i>parent_ID</i>	The ID of the operator that this node is associated with.
Child [] <i>children</i>	An array comprising this nodes children (gate inputs).
Integer <i>num_busy_children</i>	A node cannot be processed until all of its children nodes have been processed. <i>num_busy_children</i> must be 0 to process this node.
String <i>var_name</i>	This nodes Tseitin variable (gate output).

Table 7: The operation node data structure contains 6 members.

Child Structure	Description
Integer <i>type</i>	0 indicates an operation node. 1 indicates a leaf node.
Integer <i>ID</i>	The ID of this node.

Table 8: The child data structure contains 2 members.

6.3.5 Boolean Expression Syntax

Boolean formulas are specified in their natural format with the symbols shown in Table 9 below.

Operator	Symbol
AND	*, ()()
OR	+, ()+()
Implication	(()>>())
IFF	(()<->())
XOR	(()@())

Table 9: Boolean expressions should be formatted according the Symbol column above.

6.3.6 Building a Parse tree from a Boolean Expression

The pseudo code in Algorithm 1 outlines how a Boolean expression is evaluated and converted into a parse tree. This algorithm evaluates an expression from left to right. The algorithm works by evaluating one literal and one Boolean operator (to the right of the literal) at a time. Once a literal and operator have been gathered the Boolean parse tree is updated.

```

parse_tree_grow()
{
    while(!end_of_expression)
    {
        1. cur_var = Get next var;
        2. cur_op = Get next operation;
        3. Update parse_tree with cur_var and cur_operation;
    }
}

```

Algorithm 1: Pseudo code for generating the parse tree.

Step 3 from Algorithm 1 is expanded in Algorithm 2 below. The precedence of *cur_op* is compared to the precedence of *prev_op* (the last value of *cur_op*). If no operators have been processed (*prev_op* equals NULL) and an operation node is added to the empty parse tree; this only occurs when the first Boolean operator detected is processed. If *prev_op* and *cur_op* are the same then no new node is created; in this case the variable's parent ID is set to *prev_op*'s ID. If the precedence of *cur_op* is greater than that of *prev_op* then a new operation node is created as the child of the previous operation node. Finally, if all of *prev_op*'s children were literals then it is added to a list of ready nodes that can be processed when the CNF conversion algorithm is run.

```

update_parse_tree()
{
    1. Determine current literal's operation node and create new one if necessary
    if(prev_op == NULL)
    {
        create new_op_node;
        add new_op_node to parse_tree;
        cur_var.parent = new_op_node.ID;
    }
    else if(cur_op == prev_op)
        cur_var.parent = prev_op_node.ID
    else if(cur_op.precedence > prev_op.precedence)
    {
        create new_op_node;
        new_op_node.parent = prev_op_node.ID;
        cur_var.parent = new_op_node.ID;
    }
    else if(cur_op.precedence < prev_op.precedence)
    {
        cur_var.parent = prev_op_node.ID;
        if (prev_op.parent != NULL)
            prev_op.parent = parent_of( prev_op.parent);
        else
            create new_op_node;
        top_op_node.parent = new_op_node.ID;
    }
    2. Create leaf node and add it to parse tree. Leaf node's parent is determined by
    cur_var.parent (set above).

    3. Update Ready list if applicable.
}

```

Algorithm 2: Pseudo code for the `update_parse_tree()` function in step 3 of Algorithm 1.

6.3.7 Building a CNF Expression from a Parse Tree

Once the parse tree has been constructed the ready list will indicate that all of the operation nodes whose children only consist of literals are ready to be processed. It does not matter what order ready operations are processed in. Processing an operation node requires the following steps (also illustrated as pseudo code in Algorithm 3):

1. Get the operation node from the ready list.
2. Gather all of the Tseitin ID's for that operation nodes children.
3. Generate the Tseitin expression for the equivalent logic gate (according to Table 4 and Table 5).
4. If the operation node that has just been processed has a parent operation node then decrement the parent's variable indicating the number of children that have yet to be processed. If this operation node was the last thing that the parent was dependent on then add the parent to the ready list so that it can be processed.

```

generate_CNF()
{
    while(ready_list[].length() != 0)
    {
        op = pop(ready_list[]);
        children[] = gather_children_ids(op);
        op.tseitin_id = tseitin_conversion(op, children[]);

        op.parent.num_busy_children--;
        if(op.parent.num_busy_children == 0)
            push(ready_list[], op)
    }
}

```

Algorithm 3: Pseudo code for generating the CNF expression from the parse tree representation of the input Boolean formula.

6.3.7.1 Support for Variables

The Boolean to CNF compiler can process sub-expressions and store them as variables. It achieves this by storing a sub-expression's output Tseitin variable in a look up table that maps literals to their Tseitin number. This lookup table (implemented as a hash table) must be maintained regardless of variable support because variables can be repeated in any Boolean expression. A variable must be defined before it is used in another Boolean expression. Variables are simply defined by placing the sub-expression that they represent on the right side of an equals sign and their name on the left side of the equals

sign. Boolean Input 1 and Boolean Input 2 (shown below) represent identical logic but the Boolean to CNF compiler will return a more compact CNF representation for Boolean Input 1 than Boolean Input 2.

Boolean Input 1:
var=A*B
(var+C)(var*D)

Boolean Input 2:
(A*B+C)(A*B*D)

Chapter 7 - Experimental Results

7.1 System Specifications

All experiments were performed on a PC built with an Intel Core2 Duo T5800 processor (2GHz) and 4GB of system memory. The Perl scripts and MiniSat/MiniSat+ were run on Cygwin version 1.7.1 within the Windows Vista Home Premium 64-bit operating system.

7.2 Satisfiability and Pseudo-Boolean ILP Performance Results

The size of the Boolean formulation is affected by 3 main factors:

1. The number of nets specified.
2. The maximum number of two-terminal subnets, T , assigned to a net.
3. The size of the 2-dimensional grid

The tables that follow summarize the SAT solver and pseudo-Boolean ILP run time results. The tables were compiled by generating random grid files. The grid files were generated in a way that guarantees connectivity between all nets (each net was allocated a portion of the grid meaning that no wires had to cross). Each row of data in a table corresponds to a specific cell arrangement problem. The data in each row was obtained by generating 3 random grid files that represent the specific cell arrangement problem and averaging their results.

In the tables R denotes the number of rows in the grid and C denotes the number of columns. *Edges* indicates the number of virtual edges that are generated for each two-terminal subnet; this number is computed using the following equation:

$$Edges = (R-1)*C+(C-1)*R+2*(R-1)*(C-1)$$

Nets indicates the number of nets specified in the grid file. *T* is the maximum number of two-terminal nets assigned to a net. *Vars* is the number of variables defined in the CNF file. *Clauses* is the number of clauses defined in the CNF file. *Boolean Generation* is the time in seconds that it took `qca_bool.pl` to generate the bool file. *CNF Convert* is the time in seconds that it took `cnf_gen.pl` to generate the CNF file. *Minisat* is the time in seconds that it took Minisat to return a solution. *OPB Convert* is the time in seconds to convert the CNF file to an OPB file and apply the minimization constraint. *Minisat+* is the time in seconds that it took Minisat+ to return a solution.

7.2.1 Runtime vs. The Number of Nets Specified

Table 10 summarizes the runtime results as the number of the nets specified in the grid file is increase. The graph in Figure 9 shows that the number of nets has a very small effect on the size of the logic formulation; this is because increasing the number of nets only affects the Net ID vector. The Net ID vector grows logarithmically with respect to the number of nets specified. The runtime for the pseudo-Boolean ILP solver seems to unexpectedly decrease as more nets are specified.

R	C	Edges	Nets	T	Vars	Clauses	Boolean Generation	CNF Convert	MiniSat	OPB Convert	MiniSat+
20	20	1482	1	1	28909	141835	2.24	26.73	0.74	4.33	40.52
20	20	1482	3	1	32229	151811	2.06	27.85	1.28	2.96	25.20
20	20	1482	5	1	34064	158825	2.13	30.34	1.54	5.12	13.56
20	20	1482	7	1	34006	158307	2.17	30.61	1.22	4.96	14.49

Table 10: The number of nets is varied on a 20x20 grid.

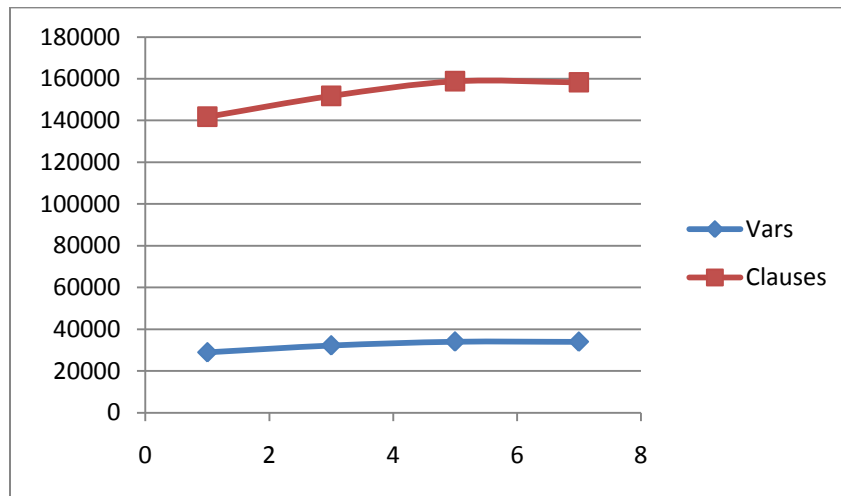


Figure 30: The number of variables and clauses generated vs. the number of nets specified.

7.2.2 Runtime vs. T

Table 11 summarizes the runtime results as the maximum number of two-terminal subnets, T , increases. The *Vars* and *Clauses* columns were actually slightly smaller for the pseudo-Boolean formulation because the minimization constraint allows constraint 9 to be eliminated; the difference was negligible. For every additional subnet specified an entire extra set of valid bits must be specified. In addition, the number of enumerations to describe constraint 7 grows exponentially. The number of enumerations to describe constraint 8 grows linearly. It was difficult to achieve valid cell arrangements for nets comprising more than 6 subnets; the small grid became too cluttered and the run times started to become very long. Figure 31 demonstrates the exponential growth of the logic formulation realized by increasing the T .

R	C	Edges	Nets	T	Vars	Clauses	Boolean Generation	CNF Convert	MiniSat	OPB Convert	MiniSat+
10	10	342	1	1	6406	30552	0.21	2.28	0.12	0.91	0.73
10	10	342	1	2	13781	65265	0.27	5.50	0.29	1.80	2.07
10	10	342	1	3	22667	105857	0.42	11.76	0.41	3.11	9.82
10	10	342	1	4	33711	154662	0.53	28.20	2.51	4.45	27.49
10	10	342	1	5	46818	210705	0.72	47.35	3.52	6.03	96.44
10	10	342	1	6	62207	276170	0.97	79.89	2.74	7.97	177.68
10	10	342	1	7	79493	347434	1.35	118.40	1.83	13.36	588.88

Table 11: The number of subnets is varied on a 10x10 grid.

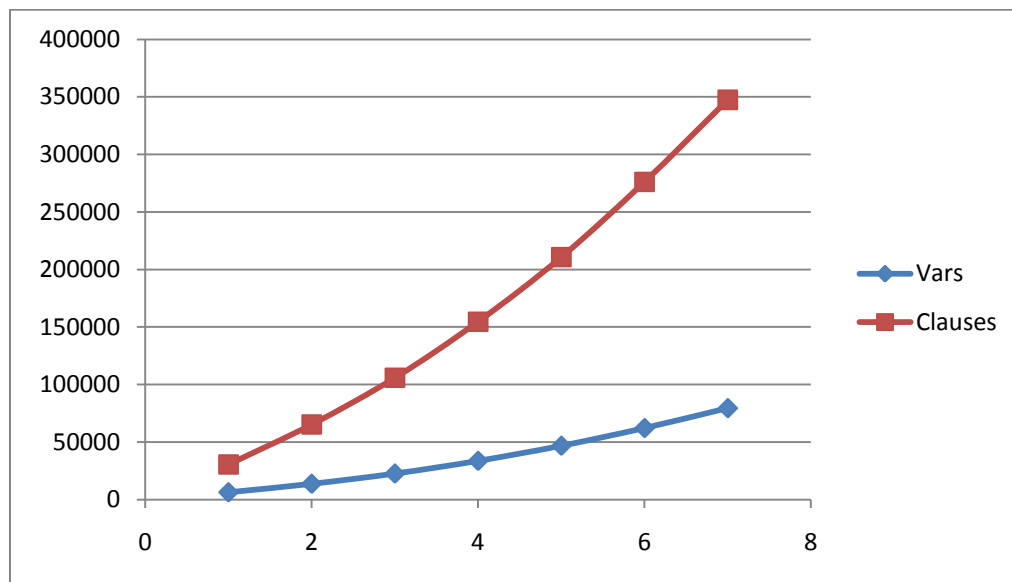


Figure 31: Number of variables and clauses generated vs. T .

7.2.3 Runtime vs. Grid Size

Table 12 summarizes the runtime results as the maximum number of two-terminal subnets, With the exception of the terminal constraints, each constraint is applied to every cell; therefore doubling the area of the grid should result in a logic formulation twice the size. Figure 32 demonstrates the linear growth of the logic formulation realized by increasing the grid size. The largest grid that could be reasonably processed by the SAT solver and pseudo-Boolean ILP solver had 30 rows and 30 columns.

R	C	Edges	Nets	T	Vars	Clauses	Boolean Generation	CNF Convert	OPB Convert	MiniSat+
10	10	342	1	1	6406	30552	0.27	2.24	0.91	0.73
20	20	1482	1	1	28909	141835	2.24	26.73	4.33	40.52
30	30	3422	1	1	67623	334634	13.94	123.00	12.35	1595.54

Table 12: The grid dimensions are varied.

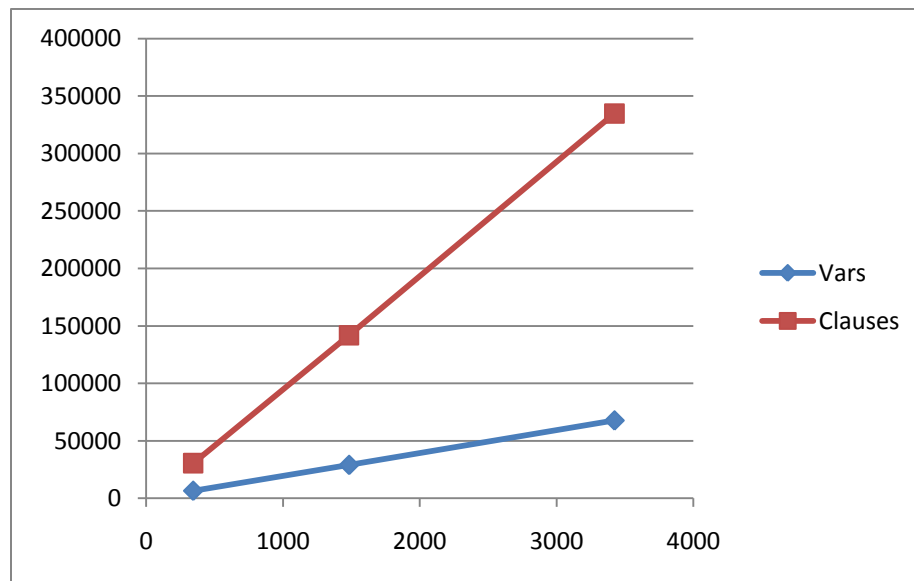


Figure 32: Number of variables and clauses generated vs. *virtual edges*.

Chapter 8 - Conclusions

This thesis successfully developed an accurate logic formulation specific to the 90 degree interconnect for the QCA cell arrangement problem. Constraints 1-3 and constraints 5-6 were based on ideas developed by Hung et al. but would have been impossible to apply without the abstraction of virtual edges created in section 4.1. Constraint 4 was created to restrict QCA cell arrangements so that the virtual edge abstraction could be fully realized. Constraints 7 and 8 were developed to prevent ambiguous QCA interconnects; these constraints could be applied to the 3-dimensional routing formulation developed by Hung et al. to achieve “cleaner” traditional electronics routings. An optimization constraint to achieve minimum length QCA routings and eliminate random wire loops inherent to the formulation developed by Hung et al. was created. The minimization constraint could also be applied to the 3-dimensional routing problem for traditional electronics to realize its benefits as described in this paper.

The logic formulation was empirically verified for correctness by applying it to actual QCA cell arrangement problems with a SAT solver. The SAT formulation was converted into a pseudo-Boolean formulation so that the effectiveness of the minimization constraint could be tested. The logic formulation and minimization constraint both functioned exactly as expected resulting in valid QCA cell arrangements. The formulations were a bit large resulting in long run times; it was ultimately determined that a different method would be more ideal for producing actual cell arrangements in a design environment.

Applying this formulation to the 45 degree QCA interconnect architecture is a trivial task. A more interesting problem is to support both 45 degree and 90 degree interconnects on the same grid. Furthermore, it would be very interesting to represent this formulation as a Pseudo-Boolean ILP problem and minimize the number of wire crossings. Other interesting future work would include developing an iterative algorithm that breaks the QCA cell arrangement problem into smaller portions that can better be handled by a SAT solver. It would also be important to explore verification techniques where the logic formulation developed in this paper might play an important role.

References:

- [1] Tougaw, Douglas P., and Craig S. Lent. "Logical Devices Implemented Using Quantum Cellular Automata." *Journal of Applied Physics* 75.3 (1994): 1818-825.
- [2] Ravichandran, Ramprasad, Nihal Ladiwala, Jean Nguyen, Mike Niemier, and Sung Kyu Lim. "Automatic Cell Placement for Quantum-dot Cellular Automata." *Proceedings of the 14th ACM Great Lakes Symposium on VLSI*. Great Lakes Symposium on VLSI, Boston. New York: ACM, 2004. 332-37.
- [3] Antonelli, Dominic A., Danny Z. Chen, Timothy J. Dysart, Xiaobo S. Hi, Andrew B. Kahng, Peter M. Kogge, Richard C. Murphy, and Michael T. Niemier. "Quantum-Dot Cellular Automata (QCA) Circuit Partitioning: Problem Modeling and Solutions." *Proceedings of the 41st Annual Design Automation Conference*. Annual ACM IEEE Design Automation Conference, San Diego. New York: ACM, 2004. 363-68.
- [4] U. of Notre Dame, "Quantum-dot cellular automata home," <http://www.nd.edu/~qcahome/>.
- [5] Nguyen, Jean, Ramprasad Ravichandran, Sung Kyu Lim, and Michael Thaddeus Niemier. "Global Placement for Quantum-dot Cellular Automata Based Circuits." *CERCS Technical Reports* (2003).
- [6] Ma, Xiaojun, Jing Huang, and Fabrizio Lombardi. "A Model for Computing and Energy Dissipation of Molecular QCA Devices and Circuits." *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 3.4 (2008).
- [7] Momenzadeh, Mariam, Marco Ottavi, and Fabrizio Lombardi. "Modeling QCA Defects at Molecular-level in Combinational Circuits." *Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. Defect and Fault-Tolerance in VLSI Systems, IEEE International Symposium, Monterey. Washington DC: IEEE Computer Society, 2005. 208-16.
- [8] Hung, William N.N., Xiaoyu Song, Timothy Kam, Lerong Cheng, and Guowu Yang. "Routability Checking for Three-Dimensional Architectures." *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS* 12.12 (2004): 1371-374.
- [9] "International Technology Roadmap for Semiconductors," <http://public.itrs.net/>.
- [10] C. S. Lent and P. D. Tougaw, "A device architecture for computing with quantum dots," *Proc. IEEE*, vol. 85, no. 4, pp. 541–557, April 1997.
- [11] C. S. Lent, B. Isaksen, and M. Lieberman, "Molecular quantum-dot cellular automata," *J. Amer. Chem. Soc.*, vol. 125, no. 4, pp. 1056–1063, January 2003.

- [12] Marques-Silva, João P., and Karem A. Sakallah. "Boolean satisfiability in electronic design automation." *Proceedings of the 37th Annual Design Automation Conference*. Annual ACM IEEE Design Automation Conference, Los Angeles, California, United States. New York: ACM, 2000. 675-80.
- [13] Sagahyroon, Assim, and Fadi A. Aloul. "Using SAT-based techniques in power estimation." *Microelectronics Journal* 38 (2007): 706-15.
- [14] Hung, William N. N., Changjian Gao, Xiaoyu Song, and Dan Hammerstrom. "Defect-Tolerant CMOL Cell Assignment via Satisfiability." *IEEE SENSORS JOURNAL* 8.6 (2008): 823-30.
- [15] Aloul, Fadi A., Arathi Ramani, Igor L. Markov, and Karem Sakallah. "Solving Difficult SAT Instances in the Presence of Symmetry." *Proceedings of the 39th annual Design Automation Conference*. Annual ACM IEEE Design Automation Conference, Louisiana, USA, New Orleans. New York: ACM, 2002. 731-36.
- [16] Smith, Brian Stephen, and Sung Kyu Lim. "QCA Channel Routing With Wire Crossing Minimization." *Proceedings of the 15th ACM Great Lakes Symposium on VLSI*. Great Lakes Symposium on VLSI, Chicago. New York: ACM, 2005. 217-20.
- [17] Manolios, Panagiotis and Daron Vroon. "Efficient Circuit to CNF Conversion." 2007. 4-9.
- [18] Parkes, Andrew J. "PBLIB - The Pseudo-Boolean Library." <http://www.cirl.uoregon.edu/PBLIB/>.
- [19] Perkowski, Marek A. "Quantum Dots and Quantum Dot Cellular Automata." http://web.cecs.pdx.edu/~mperkows/CLASS_573/MARCH_3_06/D_0021_quantum_dots.ppt
- [20] Eén, Niklas, and Niklas Sörensson. *MiniSat*. <<http://minisat.se/Main.html>>.
- [21] Jackson, Paul, and Daniel Sheridan. "Clause Form Conversions for Boolean Circuits." *SAT Selected Papers*. 2004. 183-198.
- [22] Li, Ruiming, Dian Zhou, and Donglei Du. "Satisfiability and Integer Programming as Complementary Tools." *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*. Asia and South Pacific Design Automation Conference, Yokohama. Piscataway: IEEE, 2004. 879-82.
- [23] Chambers, Benjamin, Panagiotis Manolios, and Daron Vroon. "Faster SAT solving with better CNF generation." 2009. 1590-1595.

- [24] Strichman, Ofer. "Heuristics for Efficient SAT Solving." <ie.technion.ac.il/~ofers/courses/verification/C8.0_SAT_full.ppt>.
- [25] Gupta, Pallav, Niraj K. Jha, and Loganathan Lingappan. "A Test Generation Framework for Quantum Cellular Automata Circuits." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 15.1 (2007): 24-36.
- [26] M. B. Tahoori, J. Huang, M. Momenzadeh, and F. Lombardi, "Testing of quantum cellular automata," *IEEE Trans. Nanotechnol.*, vol. 3, no. 4, pp. 432-442, Dec. 2004.
- [27] Beckett, Paul, and Andrew Jennings. "Towards Nanocomputer Architecture." *Proceedings of the Seventh Asia-Pacific Conference on Computer Systems Architecture*. Conferences in Research and Practice in Information Technology Series, Melbourne. Vol. 19. Darlinghurst: Australian Computer Society, 2002. 141-50.
- [28] K. Walus, G. Schulhof, and G. A. Jullien, "High level exploration of quantum-dot cellular automata," in *Proc. Conf. Signals, Syst., Comput.*, 2004, pp. 7–10.