國立交通大學

資訊科學與工程研究所

碩士論文

最小交點式的量子點細胞元自動化布局合成

Minimum-Crossing Layout Synthesis for Quantum-Dot Cellular Automata (QCA)

研究生：賴建丞

指導教授：李毅郎博士

中華民國九十四年十一月

最小交點式的量子點細胞元自動化布局合成

Minimum-Crossing Layout Synthesis forQuantum-Dot Cellular Automata (QCA)

研究生: 賴建丞　　　Student : Chan-cheng Lai

指導教授: 李毅郎博士　　Advisor : Dr. Yih-Lang Li

國立交通大學

資訊科學研究所

碩士論文

A Thesis

Submitted to Institute of Computer and Engineering

College of Computer Science

National Chiao Tung University

In partial fulfillment of the Requirements

For the Defgree of

Master

in

Computer and Information Science

Nov 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年十一月

最小交點式的量子點細胞元自動化布局合成

研究生 : 賴建丞　　　指導教授: 李毅郎 博士

國立交通大學 資訊科學與工程研究所

摘要

量子點細胞元自動化是一種新式奈米層級上的計算機制，它可以利用在分子上電子的表面配置來表示二元資訊。一個量子點細胞元自動化實體設計流程包含了四個步驟：切割、布置、接頭分派以及隧道繞路。因為在量子點細胞元自動化布局上的線路交點數目會嚴重影響整個量子點細胞元自動化布局設計的複雜度，這篇論文的重點將放在減少線路交點的佈局合成。在這篇論文裡，量子點細胞元自動化布局裡的布置問題將會映射到一個有名的問題「多層雙向圖交點最小化」。而為了解這個問題，我們提出了一個新的啟發示教育法。接頭分派這個步驟在隧道繞路之前，其用途是為了提供一個合法的接頭分派方式以便接下來的隧道繞路可以無礙的完成。最後，在隧道繞路這個步驟裡將提出一個名為『打破循環』的演算法，同樣是為了減少布局上的交點。基於我們的實驗數據，在布置和隧道繞路這些步驟裡，交點都有顯著的減少程度。我們利用量子點細胞元自動化設計者 2.0.3 來模擬驗證我們合成出來的布局線路。在我們的實驗裡，有一些標準線路已經驗證無誤，但有一些標準線路因為硬體的限制而無法驗證完成。

Minimum-Crossing Layout Synthesis for Quantum-Dot Cellular Automata (QCA)

Student: Chan-Cheng Lai Advisor: Dr. Yih-Lang Li

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

Quantum-dot cellular automata (QCA) is a novel nano-scale computing mechanism that can represent binary information based on spatial distribution of electron charge configuration in molecules. A QCA physical synthesis flow consists of four stages: partitioning, placement, pin-assignment and channel routing. Because wire crossings in QCA layout increase the complexity of circuit layout design, this work focus on minimizing wire crossings of the circuit under synthesis.   In this paper, the problem of QCA placement is mapped to a famous problem "*k-layer bigraph crossing problem*" and a new heuristic is developed for this problem. Pin assignment stage is prior to channel routing stage, which provides a legal pin assignment for the following channel routing stage. Finally, a new cycle breaking algorithm to reduce wire crossings in channel routing stage is presented. Based on our experimental results, placement and cycle breaking obtain good crossing reduction. We also simulate our circuit by QCA Design 2.0.3 and obtain correct simulation result and other benchmark circuits does not have simulation result since they are too large to complete simulation in time.

誌謝

我衷心的感謝我的指導教授 ： 李毅郎博士，感謝他這兩年多來耐心的指導以及在這次研究上多次深入的討論。因為教授他的啟發才讓我能順利完成這篇論文。同時我也要感謝我們實驗室的同學和學長：建毅、新育、文彬、穎樹、鵬洋和孟欣，因為他們的協助，讓我的研究順利了許多。也謝謝室友們：士政、琮義以及欣泓的幫忙，謝謝你們在我研究發生瓶頸時給予的提攜。也感謝 QCADesigner 的作者群們，你們的意見對我很有幫助，感謝你們。

最後，我要將這篇論文的完成歸功與一直在我背後支持我的家人，因為他們長期的耐心和鼓勵，才有這篇論文的出現。

Contents

List of Figures

List of Tables

Chapter 1
Introduction

The progress of CMOS technology has already brought the spectacular success of microelectronics industry. Traditional advancements of CMOS technology are always based on modifying and scaling paradigm and encourage magnitude of feature size and packing density to change rapidly over time. The number of devices integrated on a single die doubles about 18 months according to Moore's law. Since 1965, the manufacturing of integrated circuits has been governed by this scaling speed and as early as 2012, the physical scaling limit of CMOS transistor may be reached and future aggressive packing of device will be impossible. If advancement in chip performance dominates further miniaturization, we must escape from the notation of current architectures and learn to develop new paradigms even though current standard paradigm has been significantly improved by recent advances, and molecular transistor may be practical in the near future.

One such paradigms [1], Quantum-dot cellular automata(QCA) , first proposed in the early 1990s , which is followed by other studies in [2-5] A cell composed of four quantum "dots" encodes useful binary information by its charge configuration. A dot is simply a region where charge is localized. Each cell has two extra mobile electrons located at diagonal position and the other two empty dots are recorded as null state. The current state of a cell is influenced by the electrostatic effect of neighboring cells and results in a fixed state "1" or "0".

QCA cells composed of metallic dots operating at cryogenic temperature have already demonstrated by recent experiments. [4][6][7] Devices such as majority logic gate, QCA binary wire, and a single-bit memory have all been experimentally demonstrated. [8][9]

In order to obtain a more realistic outlook for systems in QCA, for the past seven years, system-level research component is included in QCA. There are two main goal of system-level work of QCA. One is as technology mature, in order to provide an infrastructure for more complex designs, the projections for synthesizable QCA circuits and the CMOS circuit in the context of the same system-level tasks will be compared. The other is to develop new architecture and system to help driving device development to get to work nano system faster.

In the context of the above goals, this paper proposes a physical design flow for generating computationally interesting, synthesizable QCA circuits. Conventional physical design problem usually divided into sub-problems of *partitioning*, *placement*, and *channel routing*. QCA Partitioning problem is first modeled and solved by A.Antonelli, et al. [10] Their partitioning model , each cell is labeled and distributed into different level, is similar with previous CA model [11]. They also provided problem modeling of placement and channel routing problem. Jean Nguyen, et al [12] [13] propose another problem formulation about QCA placement problem. They divided placement problem into zone placement and cell placement. Then Smith, et al followed Nguyen's result and provided the first QCA channel routing heuristic in [14]. This year, the method that eliminates all wire crossing of entire layout by duplicating logic resource is presented in [15].

In this paper, we focus on minimizing QCA wire crossings. Our problem definition of QCA physic design is mainly followed by A.Antonelli's formulation. Besides, we insert another stage — pin-assignment before channel routing. To minimize wire crossing, we transform the QCA circuit into the *k-layer bigraph crossing problem* and provide a heuristic solution for it.[16-19] Our heuristic combines an modified vision of guided breath-first search that present in [19] with adaptive insertion that proposed by [20]. Experiment shows our heuristic outperform conventional algorithm in *k-layer bigraph crossing problem* in most case, and obtains about 30% crossing improvement over conventional method. In addition, we provide a new heuristic in solving the *Weighted Minimum Feedback Edge Set Problem* of the channel routing stage. Our heuristic is finished in linear time and the wire crossing results average only 7% above the theoretical lower bound.

Our QCA layout is synthesized and simulated using QCADesigner [21] [22], a tool is capable of simulating complex QCA circuits on most standard platforms. We will show that our synthesis result is correct by comparing to the synthesis result of modelsim, which is capable of simulating CMOS design.

The remainder of this paper is organized as follows. In chapter 2, we provide a brief background to QCA technology. In chapter 3, we present our synthesis model and problem formulation of our physical design flow. In chapter 4, solutions of partitioning, placement, pin-assignment, channel routing is presented respectively. Chapter 5 presents our experimental results, and we conclude our paper in chapter 6.

Chapter 2
Preliminary

Before addressing our synthesis algorithm, it is necessary to review some of the basic property of QCA circuits. This chapter begins with a brief overview of device physics, and then considers basic logic. Clocking and the time rule will be introduced at last to ties everything together.

2.1 QCA basic

QCA is a novel nano device that stores logic states based on the configuration of electrons location. A quantum cell can be regarded as a set of four dots and those dots are positioned at the corners of a square. In addition to those quantum dots, the cell contains two mobile electrons that can mechanically tunnel between dots.

These two electrons are forced to the diagonal corner positions by Coulomb repulsion. The two possible polarization states represent logic "0" and logic "1", as shown in Figure 2-1.(This figure comes from[10]) Unlike traditional logic, information is transferred with Coulomb iteration between QCA cells and the QCA device is worked by spreading the state of one cell to the state of its neighbors. Hence in a QCA circuit, interconnection is the same as logic manipulation. Power dissipation in QCA circuits is relatively low compared with conventional CMOS circuit.

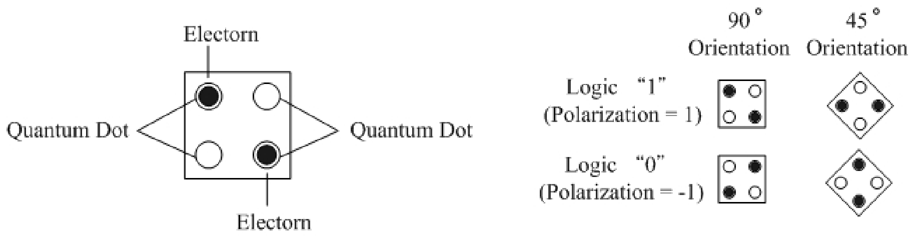


Figure 2-1 Basic QCA cell

2.2 QCA logic devices

The basic QCA logic devices include a QCA wire, QCA inverter, and QCA majority gate, will be described below.

QCA Wire: Figure 2-2 shows a QCA wire, owing to electrostatic interactions between cells, the binary information transmits from input to output. The wire propagation direction is shown in the figure. In addition to 90˚ QCA wire, QCA wire also used 45˚ QCA cell. In 45˚ QCA wire, cells alternates between the two polarizations to complete the propagation of binary information.

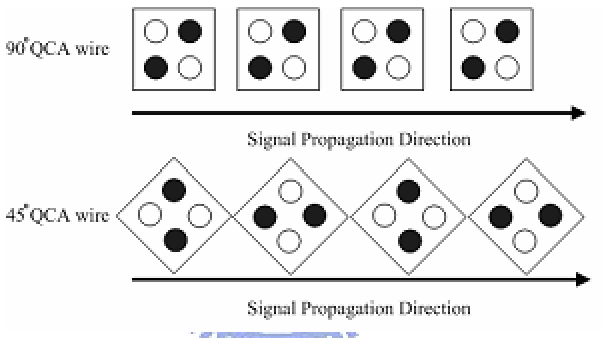


Figure 2-2 QCA Wire

QCA Inverter: A QCA inverter is shown in Figure 2-3. Cells oriented at 45˚ to its all driving cell would be resulted in opposing polarization. This phenomenon is used to create the inverter shown in the figure.

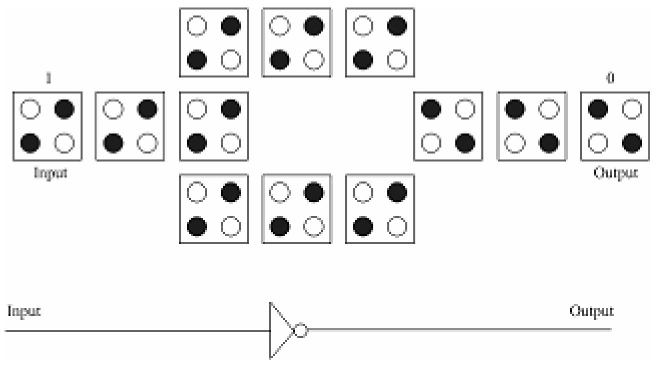


Figure 2-3 QCA Inverter

QCA Majority Gate: The function of a QCA majority gate is a three-input logic function. Assuming A, B, and C are the inputs, the logic function of this majority gate would be

$$M(A, B, C) = AB + AC + BC$$

A QCA majority gate is shown in Figure 2- 4. Computation is performed by driving the device cell to its lowest energy state, which will occur when the polarization of the three input cells is fixed.

By fixing the polarization of one input cell to the QCA majority gate as -1 or 1, and the QCA majority gate functioned as AND gate or OR gate respectively, as follows:

$$M(A, B, -1) = AB$$

$$M(A, B, 1) = A + B$$

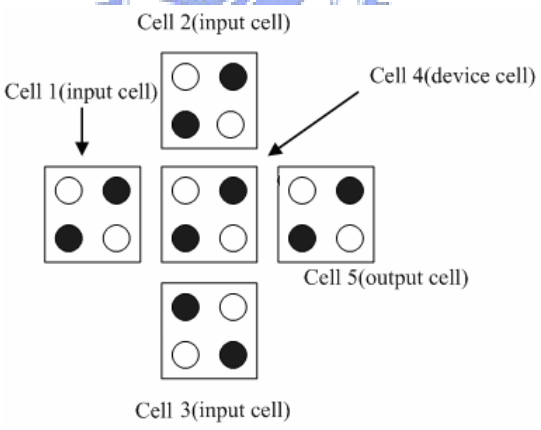


Figure 2- 4 QCA Majority Gate

QCA Wire Crossing: The traditional way to handle wire crossing in QCA layout is *coplanar wire crossing*. Figure 2 - 5 shows this way. In this example, the horizontal line is transmitting a zero and the vertical line is transmitting a one. In order to cross the line, the horizontal wire must be converted from 90˚ cells to 45˚cells. This horizontal wire of 45˚ cells can transmit information that horizontal line holds completely. When vertical line come across the horizontal wire of 45˚ cells, polarization of cell B is both influenced by cell A and cell C. Owing to polarization of cell A is stronger than that of cell C, cell B will be polarized as cell A. Coplanar wire crossing is not a robust way to handle layout with complicate wire crossings. Recently, in need of coping with complex circuits; some work has examined the multi-layer QCA. [23] On such kind of multi-layer QCA cells, vertical connection is by stacking cells on top of another. Signals can transmit to another layer where the signals can again transmit horizontally. For this reason, wire crossing can be realized in multi-layer QCA cells in the way as in Figure 2 - 6.

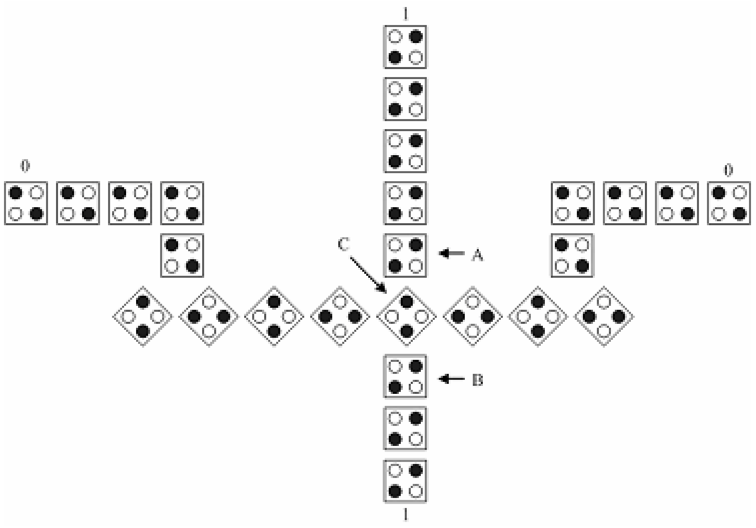


Figure 2-5 Coplanar Wire Crossing

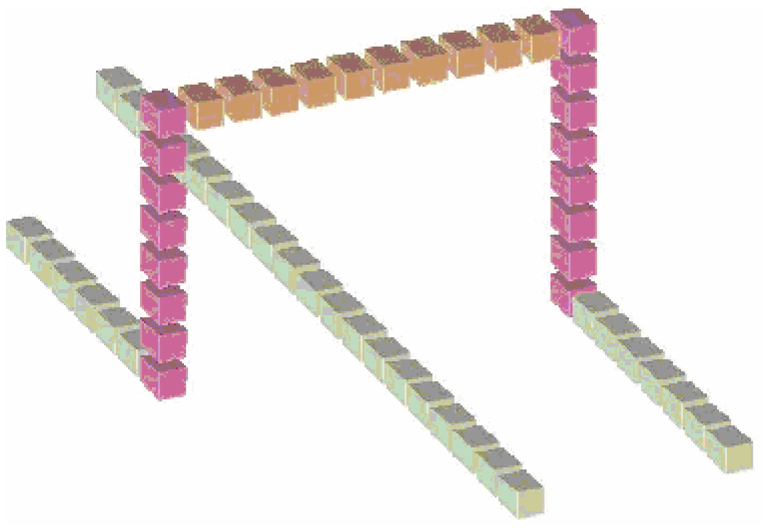


Figure2-6 QCA wire crossing realized by multilayer notation

2.3 The QCA clocking

The clock in QCA is multi-phased. It plays a key role in controlling the QCA logic functionality. To have active computation, signals must pass through *clocking zones*, which is the area where the computation is happened. These clocking zones are the successive sequence of QCA logic devices. The clocking zones create the electric field which control the lowering and rising of the potential barriers that decides the free electrons to tunnel or not. The computation between clocking zones would proceed in the sequential order. When a computation is occurring in a particular clocking zone, the clocking zone before this clocking zone must hold its cell states steady, and no computation is allowed by the clocking zone after this clocking zone.

QCA's clock was first characterized by Lent, et al. as having 4 phases as shown in Figure 2-7. During the switch phase, QCA cells begin unpolarized and their interdot electrons are in degenerate state. The electron potentials are then raised during the switch phase and the QCA cells eventually become polarized depending on the state of their input cell. In this clock phase, the actual computation (or switching) occurs. At the end of this clock phase, no electron tunneling is allowed because the electron potentials are high enough to suppress all the electron tunneling and cell states are fixed. Electron potentials are held high during the second phase, hold phase. The outputs of this clocking phase can be used as input to next clocking phase. In the release phase, electron potential is lowered and cells become unpolarized again. Cells remain in an unpolarized state during the fourth phase, relaxed phase.

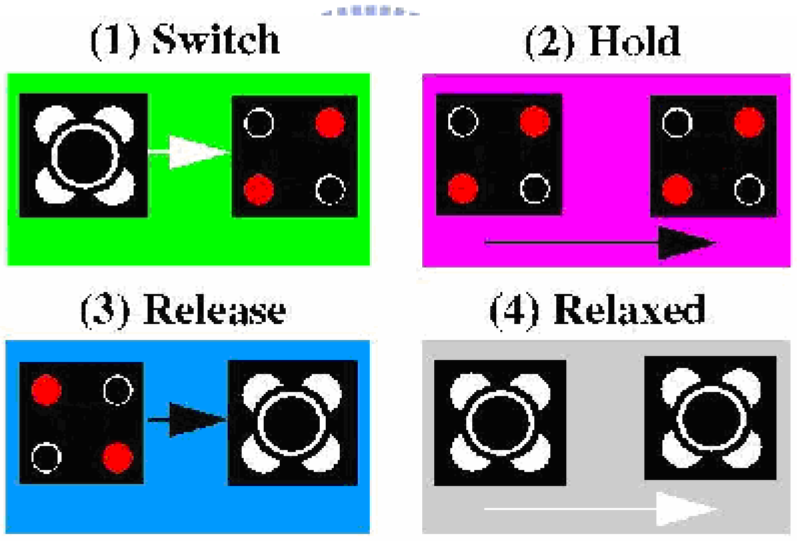


Figure 2-7. The four clocking phase of QCA

2.4 The timing rule

The timing rule of QCA circuits is strict and must be obeyed if the circuit is to function correctly. For majority gates, its input wire and output wire must be both in different clocking zones separate from the cells in the gate. Figure 2 -8 shows an example of time rule. Information transfer from one majority gate to another will cross through at least one clocking zone.

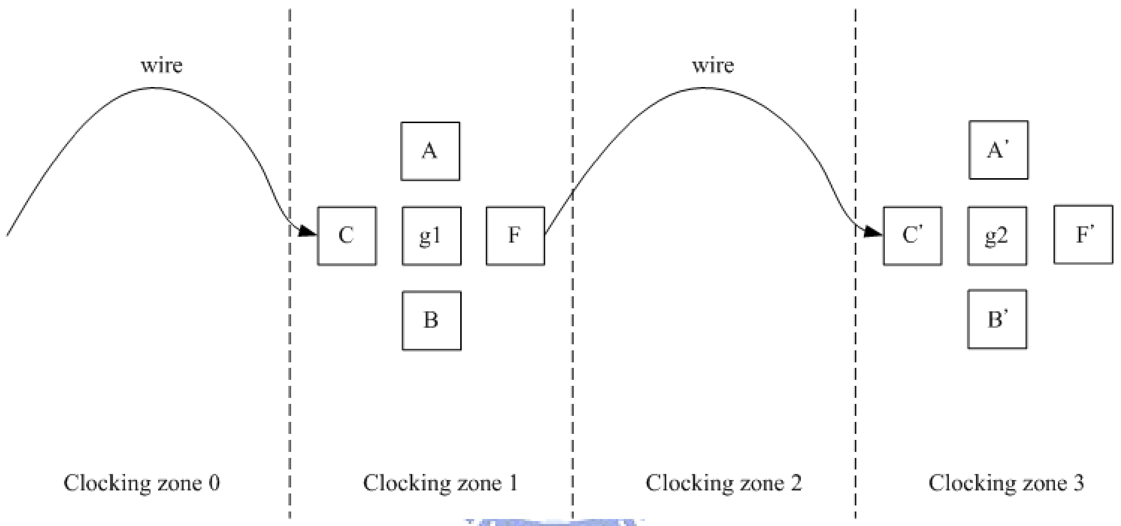


Figure 2-8 Time rule of Majority Gates

Chapter 3
Problem Formulation

Given a combinational circuit and we could represent it as a directed acyclic graph (DAG). We wish to automatically generate a QCA physical layout that realizes the circuit using a minimum number of clocking zones, meanwhile minimizing wire crossings and maximum height of overall circuit. In this chapter we first discuss our synthesis model, and in order to achieve our multiple phase objectives, we envision our QCA physical design process as consisting of partitioning, placement, pin assignment and channel routing. All steps would be formulated in this chapter later.

3.1 Synthesis Model

In order to obey time rule of QCA layout and avoid logic devices without data dependency interfere with each other. We propose our synthesis model as Figure 3-1. QCA circuit is considered as set of *levels*. One level contains two clocking zones in it, usually one for wiring, the other one for logic device. Sometimes, both clocking zones are used for wiring. A level can horizontally partition to several *rows* and vertically divide into set of *tracks*. The height of a row is as large as three cells' height plus two cell spacing. Row is used to measure the height of overall QCA circuit and all QCA wire route in the middle of row will be apart from other QCA logic devices with at least one QCA cell size. It enables every logic devices not to interfere each other. For the same season, each track is wide as three cells' width plus two cell spacing.

In our model, a QCA wire occupy one row, a QCA inverter take over two rows, and a QCA majority gate dominate three rows. We will use this measure for the following partitioning algorithm.

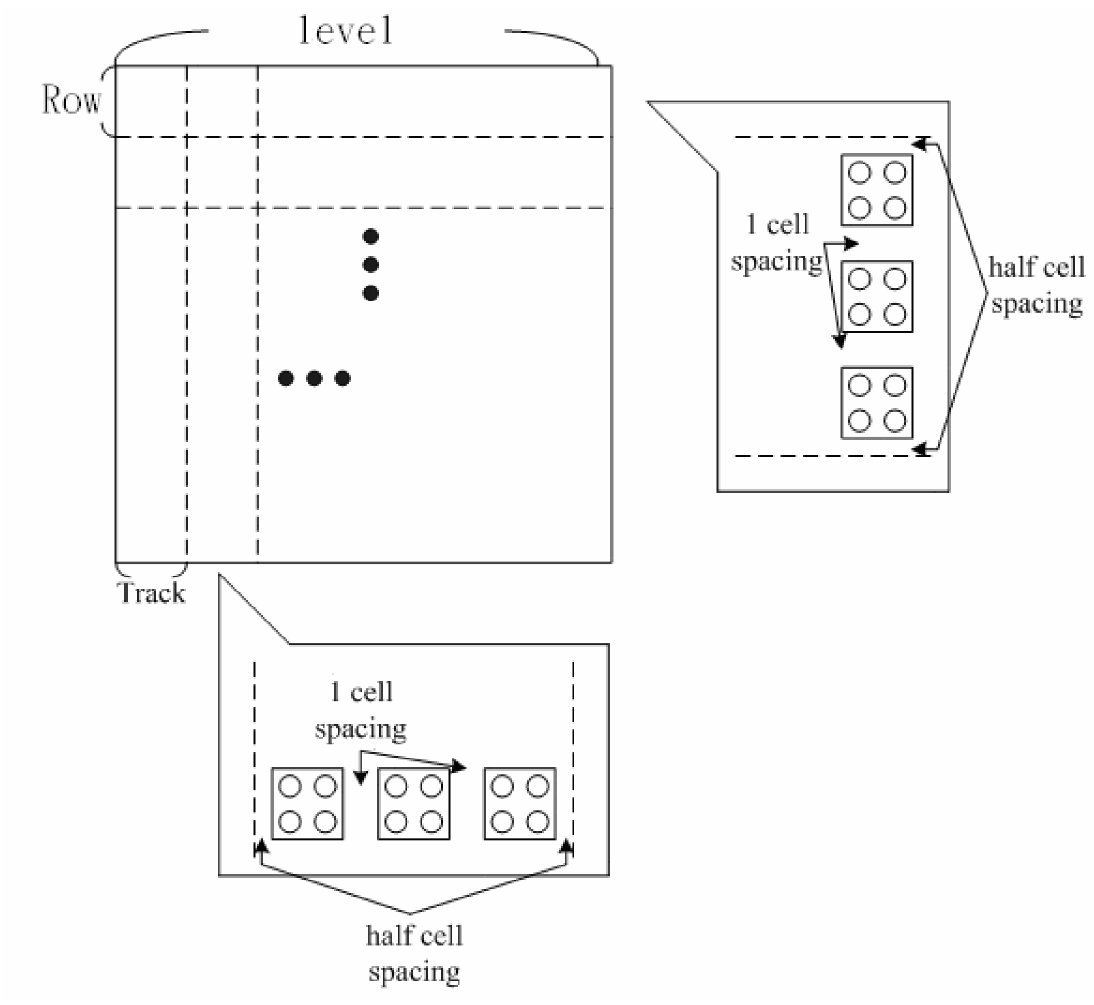


Figure 3-1 Synthesis Model

3.2 problem formulation of partitioning

This stage divides the input QCA circuit into levels which fit in with the scheduling constraint and meantime decrease the maximum level height of entire circuit.

3.2 **problem formulation of placement**

This stage rearranges the logic devices within their assigned level such that the total number of edge crossings between adjacency levels is minimized.

3.3 problem formulation of pin assignment

This stage assigns actual pin position of each logic devices of all levels which must provide a legal pin assignment for later channel routing stage while simultaneously minimize the wire length of pin-to-pin connection.

3.4 problem formulation of channel routing

This stage finishes connection among the pins in every two adjacency levels so that the total wire crossing is minimized.

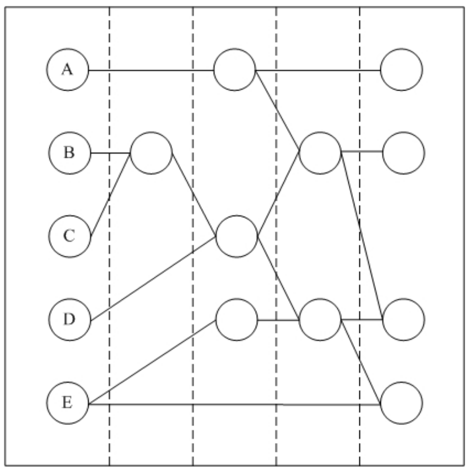
Chapter 4
Algorithm

This chapter states our total flow of QCA layout synthesis. First we partition input combinational circuit into several levels. Then we transform the circuit into the k-layer bigraph representation and focus on minimizing the total edge crossings in this graph. After that we convert this graph to physical circuit representation by pin assignment. Eventually, we implement our QCA physical layout through channel routing step.

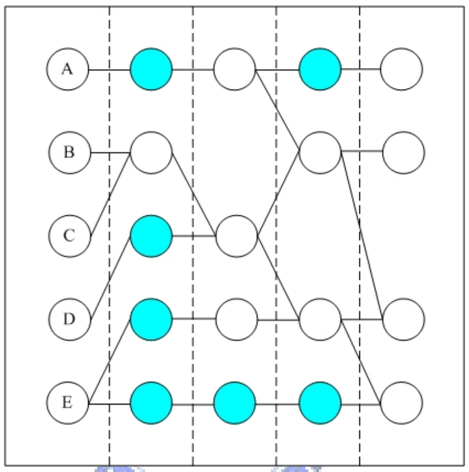
4.1 Partitioning

This section presents our partitioning algorithm, which consists of wire-block insertion, wire-block fan-out sharing, and level folding steps. In order to construct a valid schedule constraint, we do wire-block insertion that is putting each logic device into the level based on the topological ordering of these devices and then inserting wire blocks in all paths shorter than the longest *reconvergent* path. Two paths are reconvergent if they have the same starting device and the same destination device. After wire blocks are inserted into our circuit, there should be many identical wire blocks in the circuit. To remove those identical wire blocks, wire-block fan-out sharing is needed. In wire-block fan-out sharing, we merge those wire blocks which have the same input signal to maximize the sharing among the fan-outs of a logic device output.

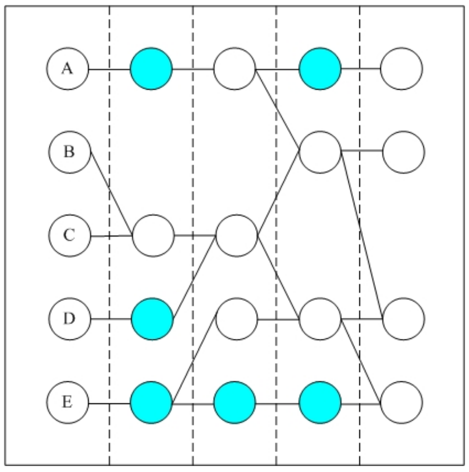
Figure 4-1(a) shows the initial circuit partition with valid scheduling. Figure 4-1(b) shows the result of wire-block insertion approach, wire blocks are those circle filled with blue. And as shown in Figure 4-1(c), two wire blocks coming from E are combined into one.



(a)



(b)



(c)

Figure 4-1. (a) An example circuit partition with valid scheduling; (b) The circuit partition after wire-block insertion approach; (c) The circuit partition after wire-block fan-out sharing approach.

The algorithm of wire-block insertion is proposed by [11], and we use this algorithm between different levels.

```
Algorithm: Wire-block Insertion
Input: A directed graph G (V, E), V is logic device and E denote data dependency
       between devices.
Begin
   n = E.pop();
   S denotes the source vertex of E;
   T denotes the destination vertex of E;
   D = level (T) – level (S);
   if ( D is bigger than one)
      Create new wire-blocks $g_1$ , $g_2$ , … , $g_{D-1}$  and add them into G ;
   for (each new wire-block)
      $g_{n-1}$ = parent($g_n$ );
      $g_n$   = child($g_{n-1}$);
      level ($g_n$) = level($g_{n-1}$) + 1;
   S = parent ($g_1$);
   T = child ($g_{D-1}$);
End
```

In this algorithm, we traverse edges one by one in the graph. For a given edge, if its two endpoints are not on the adjacent levels, a series of new wire-blocks are added between the two endpoints. These wire-blocks form a connection for the two terminals of this edge.

After wire blocks are inserted and fan-out sharing is finished, we calculate the height of each level. The heights of all levels are uniformed by level folding. We first calculate the average height of all levels and set a value of little higher than the average height as the maximum height among all levels. Therefore, the height of some levels may exceed the maximum height. We can reduce the height of those levels by folded those level into two or more levels to satisfy our maximum height constraint. This is done by inserting wire blocks in place of logic devices and placing these devices into the next level. A logic device is moved into the next new level only if replacing this device with wire blocks can decrease the level height. Figure 4-2 shows an example that move a logic device to a new level and the level height is decreased from 12 to 8. After level-folding is completed, we perform fan-out sharing again to guarantee no identical wire blocks exist.

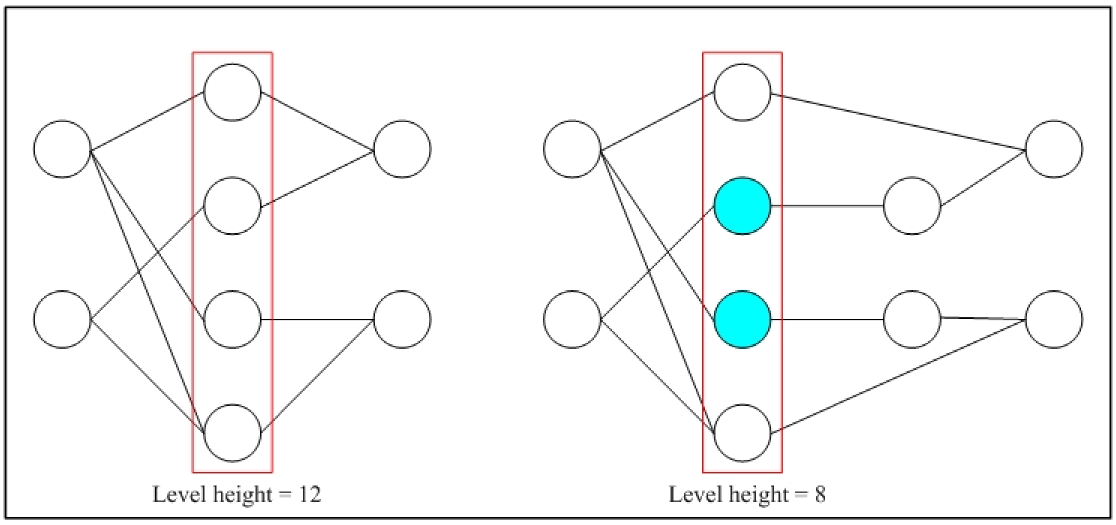


Figure 4-2. An example of level folding

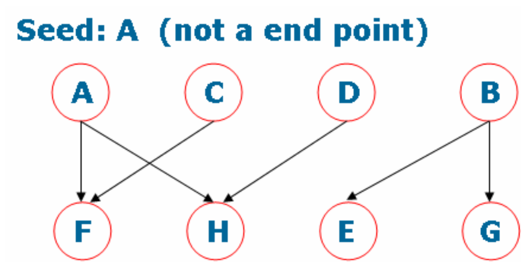
4.2 Placement

This stage reorders the logic devices level by level to minimize the wire crossing between logic devices. Placement algorithm involves *multilevel guided breadth-first search* and *adaptive insertion*. First, *multilevel guided breadth-first search* method is performed to get an initial placement. Then, placement refinement is achieved by performing adaptive insertion on each adjacent level from the last level to first level. The result of adaptive insertion is tentative and the reduction value for crossing number on each level is stored. Scanning from rightmost level toward left, a series of levels are selected for realizing placement such that this series of levels have maximum total number of crossing reduction.

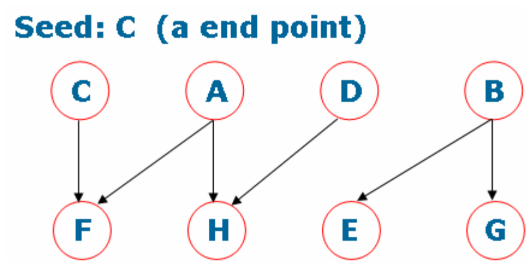
4.2.1 Guided breadth-first search

Guided breadth-first search is first proposed by [19]. The main breadth-first search is preceded by another breadth-first search whose function is to find out the longest path in the graph. After the longest path is identified, the main search begins at on end of this path and continuing attaches all other shorter paths at any branch point.

At first search, we calculate height[v], the distance form the root for each node v, and also record the depth[v], maximum value of height[u] achieved by any descendant u of v in the breadth-first search tree. Then at main search, the node s for which depth[s] is maximum will be the beginning node of this search. The reason that we select node s to be the begging node of the main search would be illustrated as Figure 4-3. In Figure 4-3(b), BFS starting from an end point of a path results in no edge crossing in the graph. This result is better than the result shown in Figure 4-3(a).   When one node k has two or more children, we traverse its children k_i by increasing order of depth [k_i] and ties are broken by traversing the node with larger height [k_i] first. We would illustrate this in Figure 4-4. In Figure 4-4(a), we complete our BFS traversal with smaller depth[v] first, and complete the traversal with larger depth[v] first in Figure 4-4(b). As shown in graph, traversing with smaller depth[v] first results in less edge crossing. While main search is finished, a series of numbers are assigned to each node based on the order of visitation in the main search. And we use these numbers to rearrange nodes on each level of the graph.

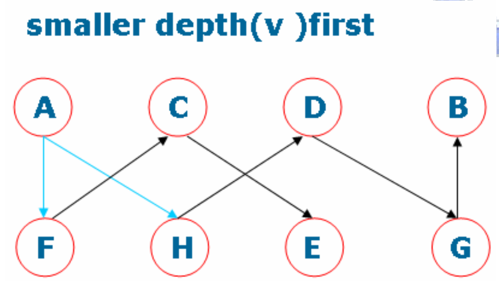


(a)

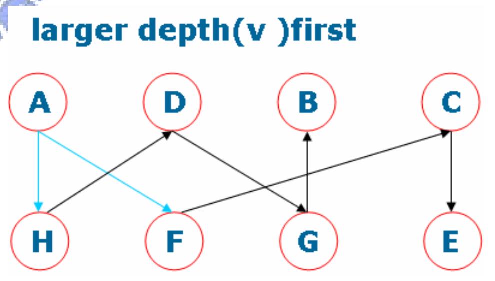


(b)

Figure 4-3. (a) BFS starting not from a end point (b) BFS starting from a end point



(a)



(b)

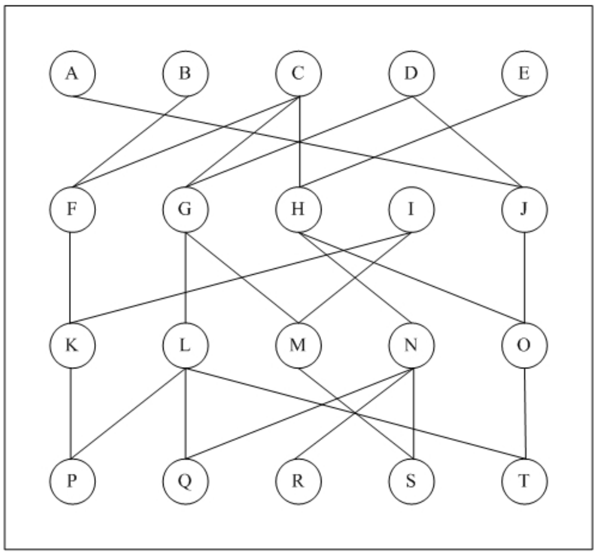
Figure 4-4. (a) BFS traversal order with smaller depth[v] first (b) BFS traversal order with larger depth[v] first

4.2.2 Multilevel guided breadth-first search

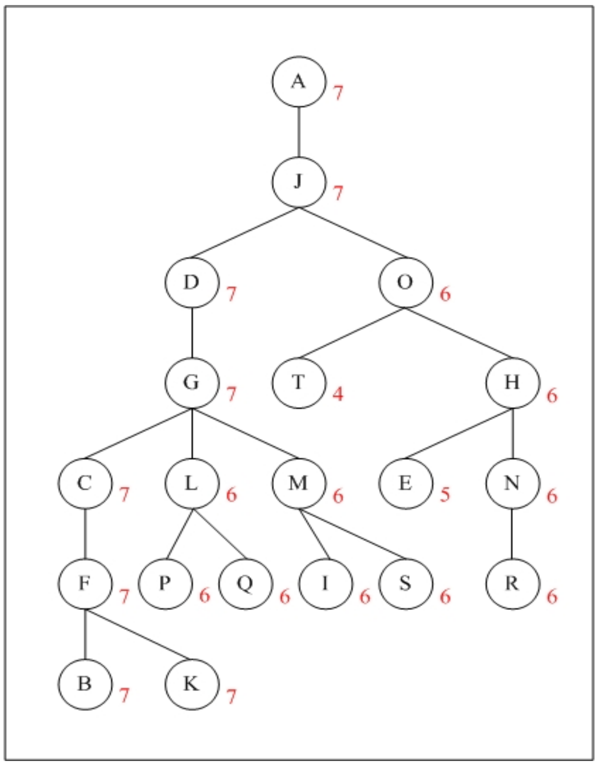
In our heuristic, we would like to minimize total offset between levels. Thus we apply guided breadth-first search to entire circuit, i.e. all logic devices are traversed in this search. In the first search, current traversed node will collect the neighbor nodes in the pre-level first and then collect nodes in the post-level. This method could provide an initial placement that the longest path (or largest component) of the graph will be decomposed from this circuit first and shorter path (or smaller component) of the graph start to attach to the largest component. And if there are several disjoint component in this graph, those components will separate from each other. We would illustrate multilevel guided breadth-first search through Figure 4-5(a) to Figure 4-5(d).

Figure 4-5(a) shows a 4-level bigraph with initial presentation. We select node A as the seed for the first breadth-first search and result is shown in Figure 4-5(b). In Figure 4-5(b), depth number is listed aside the node, for instance, depth number of node A is seven. Figure 4-5(c) shows the result of the main breadth-first search. In this graph, node L has three children T, Q, G. Because depth (T) < depth (Q) < depth (G), we traverse these nodes in the order of T, Q and G. Figure 4-5(d) is the placement of multilevel guided breadth-first search. All nodes are sorted by the increasing order of the number listed by Figure 4-5(c).

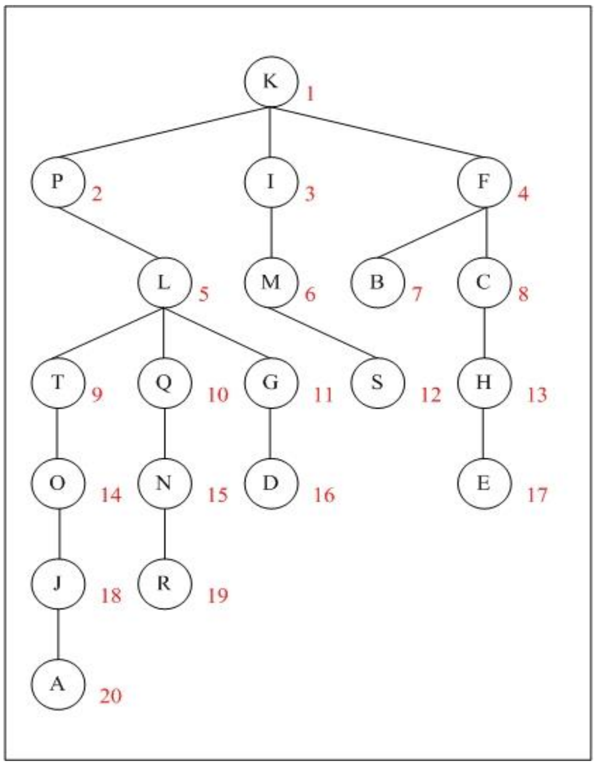
In our example of multilevel guided breadth-first search, there are 18 wire crossings in the initial presentation. After applying multilevel guided breadth-first search, only 8 wire crossings remain in the graph, in other words, 10 wire crossings is reduced in the graph after multilevel guided breadth-first search.


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T

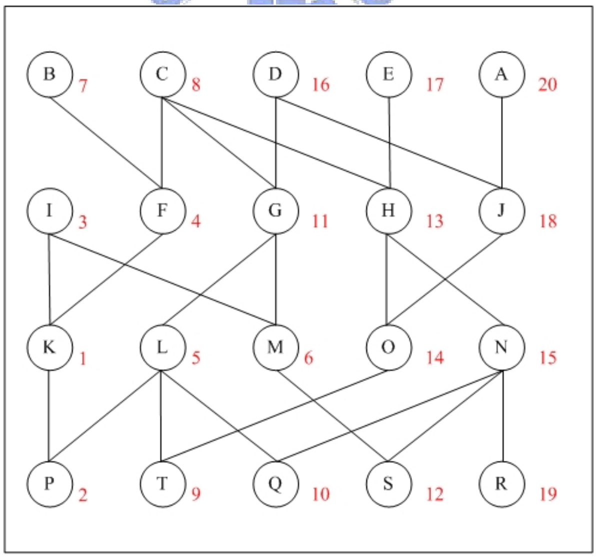
(a)


A 7
J 7
D 7
O 6
G 7
T 4
H 6
C 7
L 6
M 6
E 5
N 6
F 7
P 6
Q 6
I 6
S 6
R 6
B 7
K 7

(b)



(c)



(d)

Figure 4-5. (a) An example of the initial placement of a 4-level bigraph; (b) The first breadth-first search result of (a), number aside the node is the depth of this node; (c) The main breadth-first search result of (a), numbers aside the node is the visitation of the main search; (d) Placement after multilevel guided breadth-first search, nodes in a level is sorted by numbers in (c).

4.2.3 Adaptive Insertion method

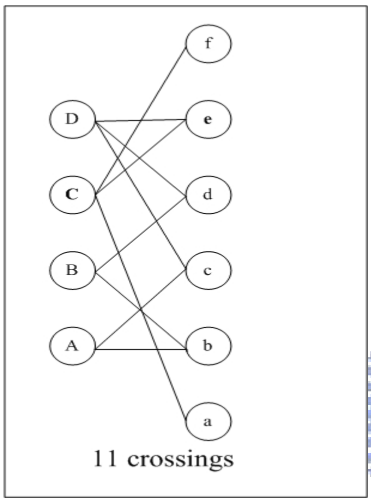
Local search [20] is a popular way to improve solutions in bigraph crossing problem. Repeat simple operation on the current ordering until no instance of the operation would improve reducing number of crossings. An example of an operation is neighbor swapping, which is swapping nodes (at position) *i* and *i* + 1 on level *l*. Such operation would be repeated until no choice of *i* could decrease the number of crossings.

Adaptive insertion is a kind of local search based on neighbor swapping in the way: each operation inserts a node at any position among other nodes on its level, and each node is inserted mostly once during a pass. Assume node *i* is inserted before node *j*, where $j < i$, The resulting cost change, $D_l(i,j)$, is $\sum_{k=j}^{i-1} D_l(i,k)$ — the effect of the insertion is that of a succession of swaps of node *i* with node *j* , *j* + 1, … , *i* -1. The condition node *i* is inserted after node *j* is similar with the resulting cost change $D_l(i,j)$ = $\sum_{k=i+1}^{j-1} D_l(i,k)$.

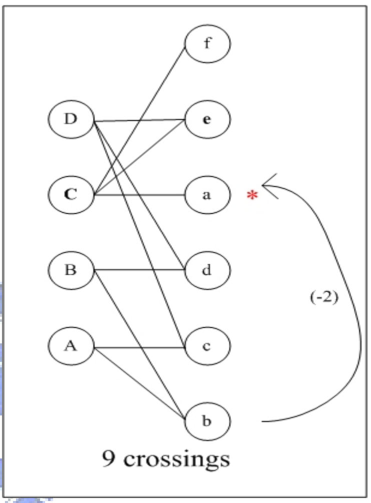
In our heuristic, one pass of adaptive insertion does a bottom-to-top sweep of logic devices on a level *l*. Devices are not allowed to stay in place even if no insertion would decrease the number of crossings. If device *i* is already inserted in a pervious operation, node *i* is *marked* and it is not selected any more during the remainder of the current pass.

To illustrate one pass of adaptive insertion, we use an example starting from Figure 4-6 (a) to perform a series of node swapping. The first node we selected to perform inserting operation is node a. Figure 4-6(b) shows the best position for node a is to insert it above node d, yielding a decrease of two. The next unmarked node is b. Node b is forced to move and finds its best position is below node d as shown in Figure 4-6(c).

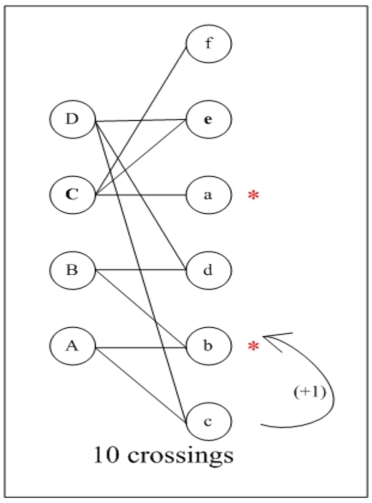
After moving node b, the total crossings number is increase by one. Next we select node c as our seed to do operation. Node c will be placed above node b and gain a crossing reduction of one as shown in Figure 4-6(d). Next operation is swapped node a and node d with no change in number of crossings. Finally node e is placed to uppermost position as shown Figure 4-6(f) with total wire crossing number of eight.



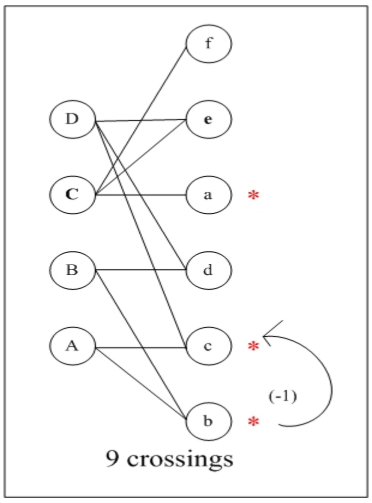
(a)



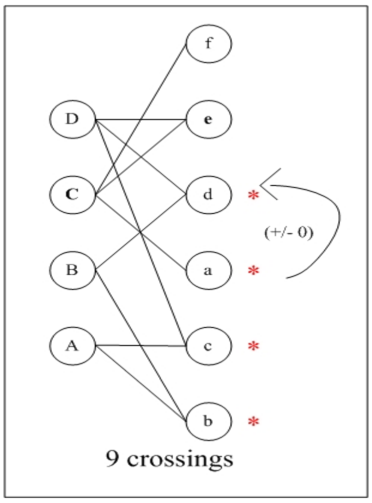
(b)

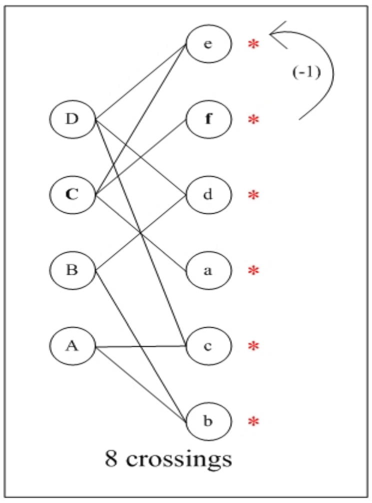


(c)



(d)





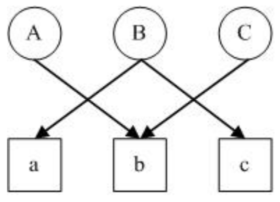
(e) (f)

Figure 4-6. Adaptive insertion on a simple example

To finish adaptive insertion is time-consuming because we should compute variations of wire crossings after every node swapping. Thus we use the adjacency matrix to compute the number of wire crossings to save the computing time.

In a bipartite graph, there is a wire crossings between two layers x and y if x_i connects to y_m , x_j connects to y_n and $x_i < x_j$, $y_m > y_n$ where i , j , m , n denotes the relative positional ordering of the nodes. In terms of an adjacency matrix, this can be considered as if point (i,j) is included in the lower left sub-matrix of (m,n) or vice versa. Therefore the total crossing is computed by adding the product of every matrix element and the sum of its left lower sub-matrix entries. Because this is very computational expensive, we implement it with the incremental wire crossing method proposed by [13] instead of computing the matrix directly. Figure 4-7 shows an instance of wire crossing computation .In this method, we firs calculate the row-wise sum of all entries as in Figure 4-7(c). Then we compute the column-wise sum of this row-wise sum matrix as in Figure 4-7(d). Finally, we calculate the sum of all the entries(r,c) in the original matrix by the entries (r + 1, c - 1) in the column-wise sum matrix to obtain the total wire crossing.

Incremental wire crossing method enables us to perform node insertion without computing wire crossing individually. In stead, we just update the value of rows after every operation to get the total number of wire crossings.



(a)

	a	b	c
A	0	1	0
B	1	0	1
C	0	1	0

(b)

	a	b	c
A	0	1	1
B	1	1	2
C	0	1	1

(c)

	a	b	c
A	1	3	4
B	1	2	3
C	0	1	1

(d)

Figure 4-7 Illustration of wire crossing computation. (a) given graph, (b) initial adjacency matrix, (c) row-wise sum, (d) column-wise sum.

4.3 Pin assignment

This section presents our pin assignment algorithm, which consists of greedy pin assignment and pseudo routing steps.

4.3.1 Greedy pin assignment

In this step, we assign the input pin positions of all device blocks from the device on topmost position of the level to the device on bottommost position of the level. Then if there are some empty rows below all device blocks in this level, we begin to move device block from bottom to top of this level to their *best position*. The best position of the device block is the position that makes wire length of pin-to-pin connection of the device minimized. Sometimes, best position of a device block is not unique. In such condition, we would shift this device block to the bottommost best position for the reason that preserve most moving space for other device blocks. If best position of a device block is already occupied by other devices, this device block would be placed to an unoccupied position that is closest to best position. We would illustrate this step from Figure 4-8(a) to Figure 4-8(g)

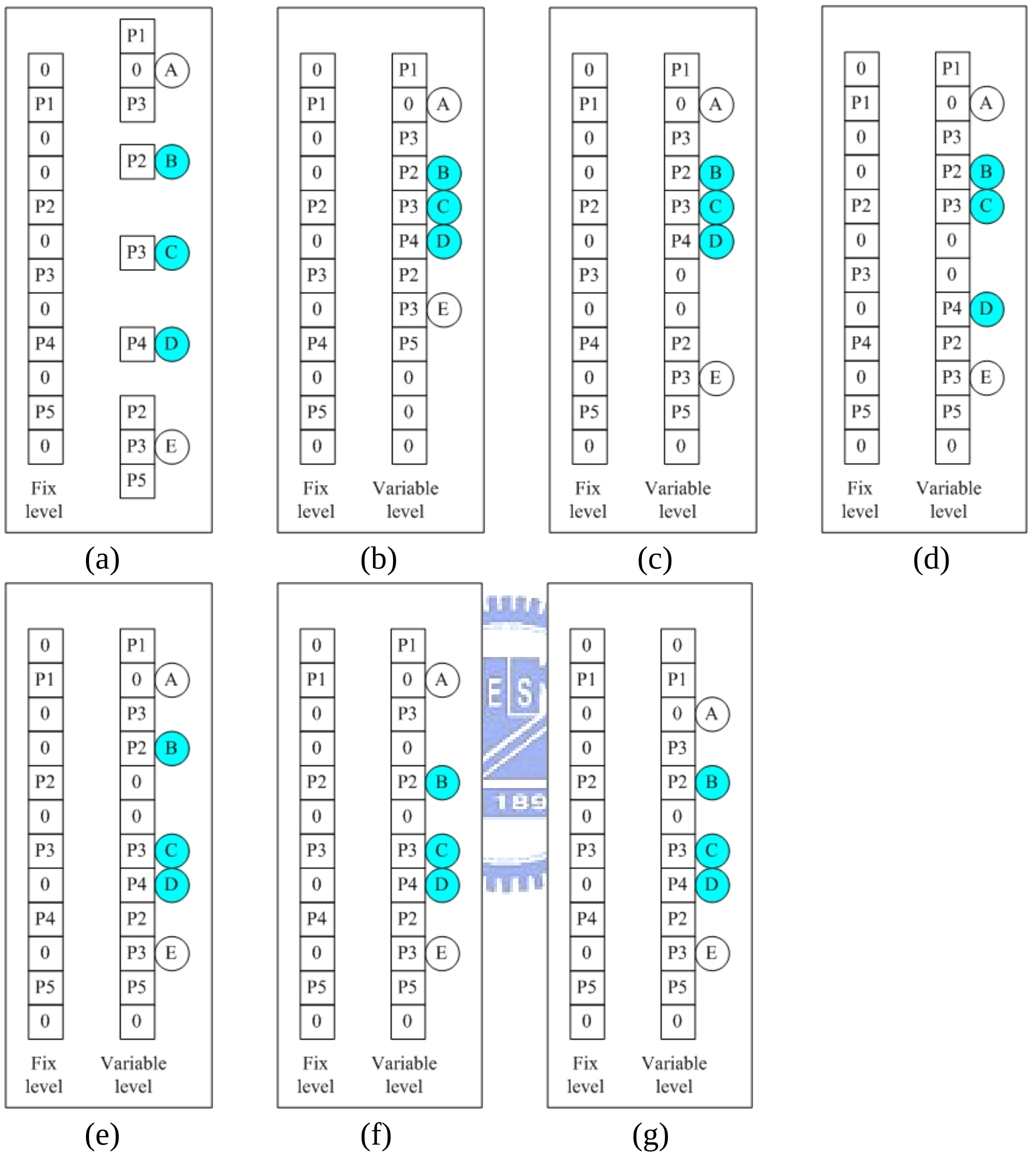


Figure 4-8. An example of greedy pin assignment

In Figure 4-8(a), the left side is the fixed level (pins are already assigned); right side is the device set of the variable level. Figure 4-8(b) shows the initial pin assignment of the variable level. Because three empty rows remain in this level, we begin to shift devices from E to A to their best position. In Figure 4-8(c), device E is shifted to its best position. Then we can notice that the best position of device D was occupied by device E already, thus we place it to the position closest to best position as shown in Figure 4-8(d). Other devices above device D is able to shift to the best position, Figure 4-8(e-g) show those results.

4.3.2 Pseudo routing

After greedy pin assignment, there may be still some *unroutable* nets. A net is unroutable if it forms a cycle in vertical constraint graph and can't resolve this cycle by doglegging. We would verify such case exists or not by pseudo routing. When an unroutable net is found, we would insert a new row into this level or slightly shift the position of pins nearby the pin of this unroutable net to make this net routable. Pseudo routing will be repeated until no unroutable nets found in our pin assignment results.

4. 4 Channel Routing

This stage will finish the wire routing inside every level. Although in our synthesis model, level is a horizontal column, in this section we would like to lie down all levels as Figure 4-9. This is because channel routing like Figure 4-9 is a well-known form of channel routing.

4.4.1 Overview

We would finish the wire routing for each level in 5 steps. First, the level pins are scanned from left to right, and the VCG is constructed (step 1). Since there can't be any cycle existed in VCG, cycles in the VCG are removed by doglegging (step 2). Once the VCG is acyclic, we would add crossing edges to VCG to reduce wire crossing (step 3). If cycles exist in the VCG, minimal weighted crossing edge set are removed and to make VCG becoming acyclic again (step 4). Eventually, we apply the LEF algorithm to assign track to each net and finish routing of this level. Channel routing process is applied to every level to implement entire circuit.

4.4.2 Doglegging

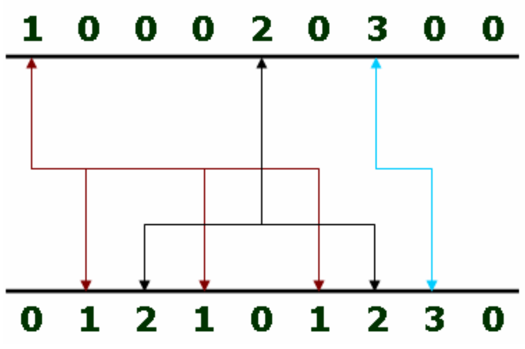
Doglegging is to split of horizontal segments of a net. This is used, not only to remove cycles in the VCG, but also used to minimize the number of horizontal tracks. We can apply DFS to determine whether VCG contains cycles or not. Once a cycle is found, the net in this cycle is divided into several subnets and a vertical dogleg is inserted. Each of these subnets is created and added into the VCG to remove these cycles.

4.4.3 Crossing edge insertion

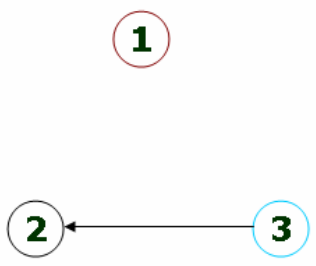
It’s clear that wire crossing can only occur between nets which overlap horizontally. And the number of wire crossings between any arbitrary pair of horizontally overlapping nets is strongly influenced by their vertical ordering.

Therefore, in order to reduce crossing, we use the notation “*crossing edges*” which is first proposed by [14], between nets in the VCG. In order to drive those horizontally overlapping nets to form the vertical relationship which results in the minimum number of crossing between them. Each crossing edge is a directed edge and assigned a weight that determines the number of wire crossings saved by placing the net that denotes the source point above the net that denotes the destination point.

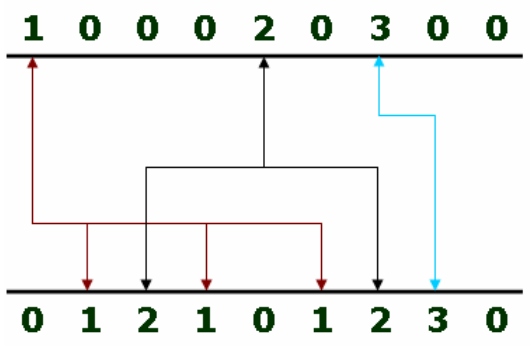
For example, consider Net 1 and Net 2, which overlap horizontally in Figure 4-9. Figure 4-9(a) shows if Net 1 places above Net 2 then they will crossover three times. But if Net 1 places below Net 2 as shown in Figure 4-9(c), there is only one crossing between them. Therefore, we modify VCG shown in Figure 4-9(b) to Figure 4-9(d). Crossing edge sources form Net 1 is weighted by two and points to Net 1. Figure 4-9(c) is the result channel routing of VCG shown in Figure 4-9(d) which is the fewest wire crossings.



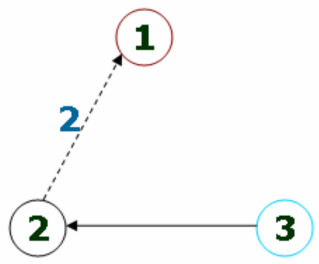
(a)



(b)



(c)



(d)

Figure 4-9. (a-b) A channel and VCG with minimum channel width. (c) Optimum solution for minimizing wire crossing. (d) Modified VCG with inserting crossing edge.

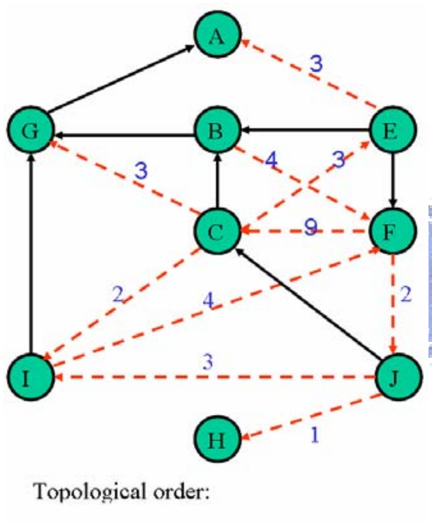
4.4.4 Cycle break

As stated in previous section, the weight of crossing edges determines the number of wire crossings reduction. Hence if VCG is acyclic after crossing edge insertion, we preserve all crossing edges to obtain a result with minimum wire crossings. Otherwise, if

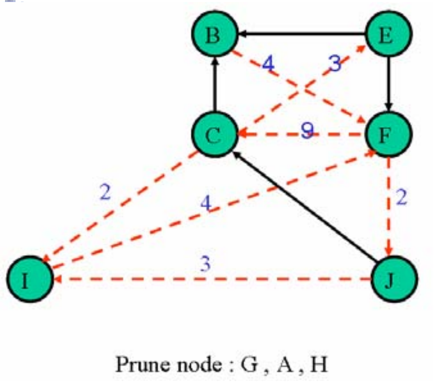
VCG is not acyclic after insert crossing edges, the cycles must be removed before track assignment begins. Because if we reserve the crossing edge set with higher total weight, the result of channel routing has fewer wire crossings. Our goal is to find a set of crossing edges $A \subset E$ with the minimum total weight such that $G - A$ is acyclic.

Since a acyclic directed graph has a topological ordering, we develop our cycle break algorithm as shown in Figure 4-11. The main idea of our algorithm is to enforce a topological ordering of VCG and remove all *violating edges* that violate this topological ordering, i.e. edge start from the vertex with larger order to the vertex with smaller order. Therefore our goal is to find a topological ordering of VCG that has the violating edges set with minimum total weight.

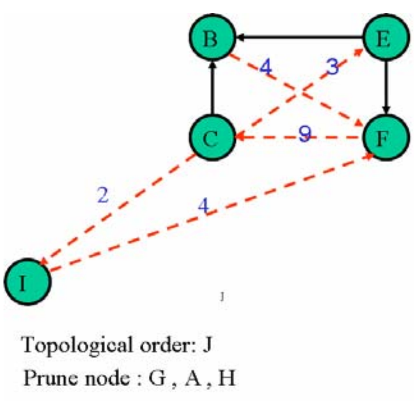
In our algorithm, we prune the vertices which have no outward edges first because these vertices wouldn't introduce any violating edges if we place them in the tail of topological ordering list. Then we calculate the cost of the *candidate vertex* that has no inward vertical constraint edge. Cost of candidate vertex determines that if we want to break a cycle, how many crossing reduction we would lose. Thus we pick the vertex with lowest cost into the front of topological ordering list and update VCG. After update VCG, if any vertices which have no outward edges exist, we would prune them as the reason stated above. When the list of topological ordering is completely formed, we start to traverse this list from the begging to record all violating edges. Because VCG is a directed graph, not all violating edges introduce a cycle; each violating edge must be verified. If this violating edge actually introduces a cycle, we remove it form VCG, otherwise we would preserve it. For example, Figure 4-10(a) is a cyclic VCG. We first prune vertex that has no outward edge in the order of H, A, G. Figure 4-10(b) shows the result after pruning. In Figure 4-10(b), vertex I, E, and J is candidate vertex and vertex J has the lowest cost of 1. Therefore, we remove J form VCG and Figure 4-10(c) shows this result. In Figure 4-10(c), vertex E has the lowest cost of 3/2, thus vertex J would be the next node removes from the graph. When all vertices are removed form the graph, a topological ordering would be developed. In Figure 4-10(d), such topological ordering is shown below the graph and violating edge is colored with blue. In this example, all violating edge actually introduces a cycle. Consequently, all violating edge must be removed. Final result is shown in Figure 4-10(e).



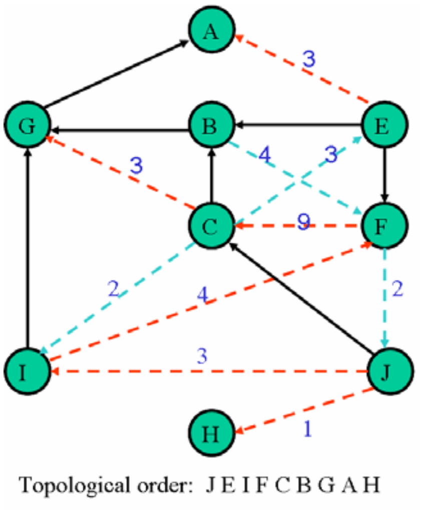
(a)



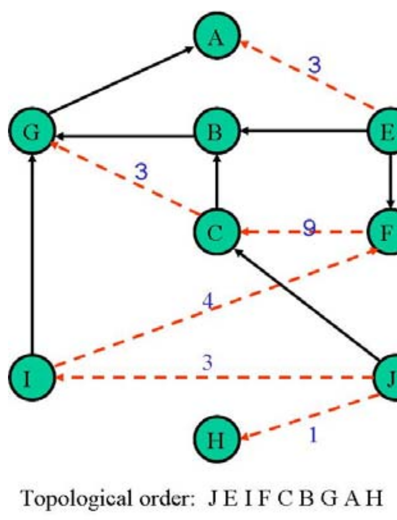
(b)



(c)



(d)



(e)

Figure 4-10. Example of cycle break algorithm

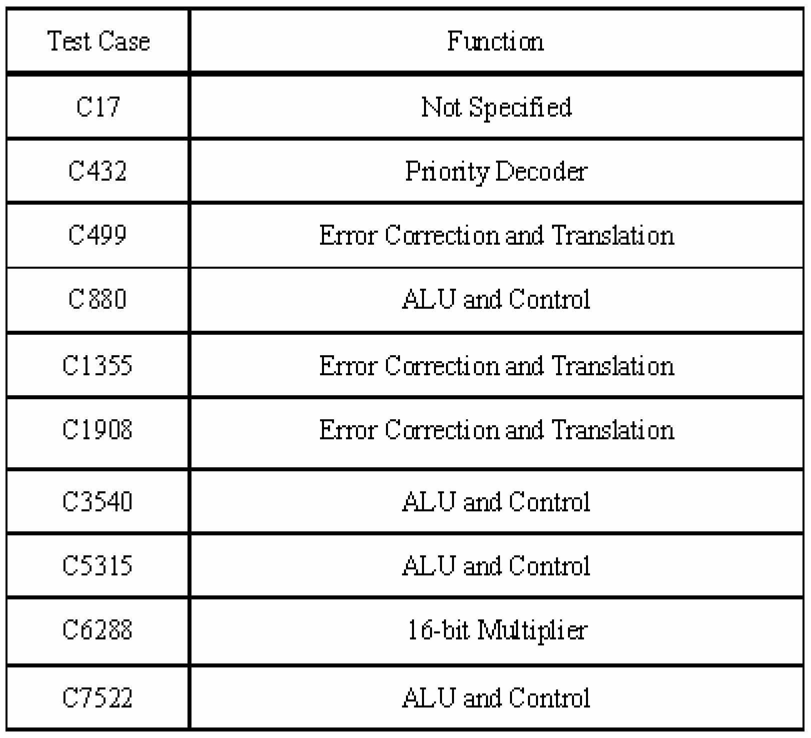
```
Algorithm: Cycle Break
Input: A cyclic directed graph G( V , E )
output: A acyclic directed graph G’ ( V ’ , E’ ) = G( V , E – A)
Begin
    Repeat
            remove all vertices of G which has no outward edges from G；
            push those vertices to a stack $S_Q$；
    Until (all vertex of G has at least one outward edge)
    Repeat
            if( vertex $V_i$ has no inward vertical constraint edge)
                $\text{Cost}(V_i) = \frac{\text{Total weight of inward crossing edges of Vi}}{\text{Number of outward crossing edges of Vi}}$
            Select a vertex $V_t$ with lowest cost , remove $V_t$ form G；
            Insert $V_t$ into a list $L_Q$；
            if( G exists vertices which has no outward edges)
            Remove those vertices from G and push them to $S_Q$；
    Until no more vertices in G
    while($S_Q$ is not empty)
            Pop vertex form $S_Q$ and insert this vertex to $L_Q$；
    All Edges in $L_Q$ that violate topological ordering of $L_Q$ and results in a cycle of
    VCG will be removed from VCG；
End
```

Figure 4-11 Cycle Break Algorithm

Chapter 5
Experimental Results

Our QCA synthesizer was implemented in C++/STL and complied with Borland C++ Builder running on 2.4 GHz Pentium 4 PC with 256 RAM. Ten combinational circuits were selected from ISCAS85 benchmark. Table 1 shows these circuit and report their functionality.

Table 1. Circuit Statistics for QCA synthesizer



Test Case	Function
C17	Not Specified
C432	Priority Decoder
C499	Error Correction and Translation
C880	ALU and Control
C1355	Error Correction and Translation
C1908	Error Correction and Translation
C3540	ALU and Control
C5315	ALU and Control
C6288	16-bit Multiplier
C7522	ALU and Control

Table 2 shows the edge crossing results after placement stage. We list our algorithm in two ways: with or without adaptive insertion and compare our results with conventional placement algorithm: barycenter ordering and median ordering. Each entry list in Table 2 is the value of total edge crossings. Our placement algorithm was outperformed conventional placement method in most cases except for C432. We conclude the memory usage and time usage of these algorithms in table 3.   In table 4, we normalize our algorithm results with results of conventional method in addition to C17 and C432.   In comparison, the method present here obtains average 27% crossing improvement over barycenter ordering and 38% crossing improvement over median ordering. Even without adopting adaptive insertion, improvement over barycenter ordering and median ordering is still obtained.

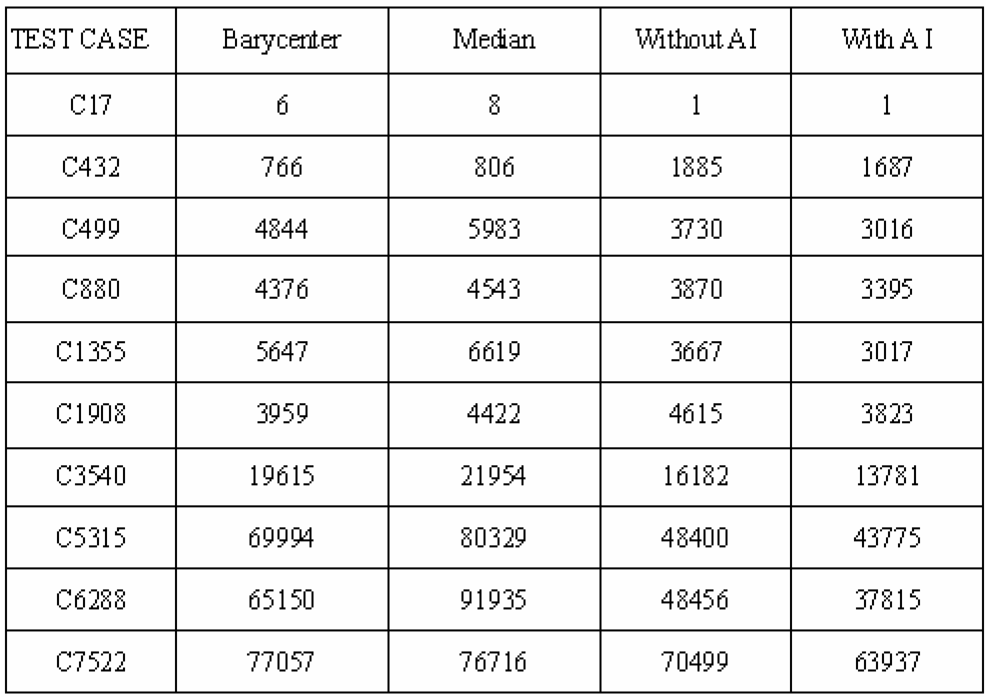


Table 2. QCA placement results

Each entry denotes total edge crossings among levels

TEST CASE	Barycenter	Median	Without A I	With A I
C17	6	8	1	1
C432	766	806	1885	1687
C499	4844	5983	3730	3016
C880	4376	4543	3870	3395
C1355	5647	6619	3667	3017
C1908	3959	4422	4615	3823
C3540	19615	21954	16182	13781
C5315	69994	80329	48400	43775
C6288	65150	91935	48456	37815
C7522	77057	76716	70499	63937

Table 3 Memory usage and time usage of algorithms of table 2
M denotes the maximum memory usage during this algorithm, and T denotes time usage of this algorithm

TEST CASE	BaryCenter		Median		Without AI		With A I	
	T(sec)	M(k)	T(sec)	M(k)	T(sec)	M(k)	T(sec)	M(k)
C17	<0.001	4	<0.001	4	<0.001	8	<0.001	8
C432	0.015	4	0.015	4	0.015	96	2.204	120
C499	0.016	4	0.016	4	0.063	128	18.574	288
C880	0.016	4	0.016	4	0.047	92	23.203	284
C1355	0.015	4	0.015	4	0.047	104	17.36	268
C1908	0.016	4	0.016	4	0.047	156	21.953	260
C3540	0.016	4	0.016	4	0.080	208	91.281	664
C5315	0.032	4	0.032	4	0.188	448	2658	1140
C6288	0.047	4	0.047	4	0.329	708	3513	2436
C7522	0.032	4	0.032	4	0.265	960	4457	1732

Table 4

Compare our placement algorithm with conventional placement algorithm

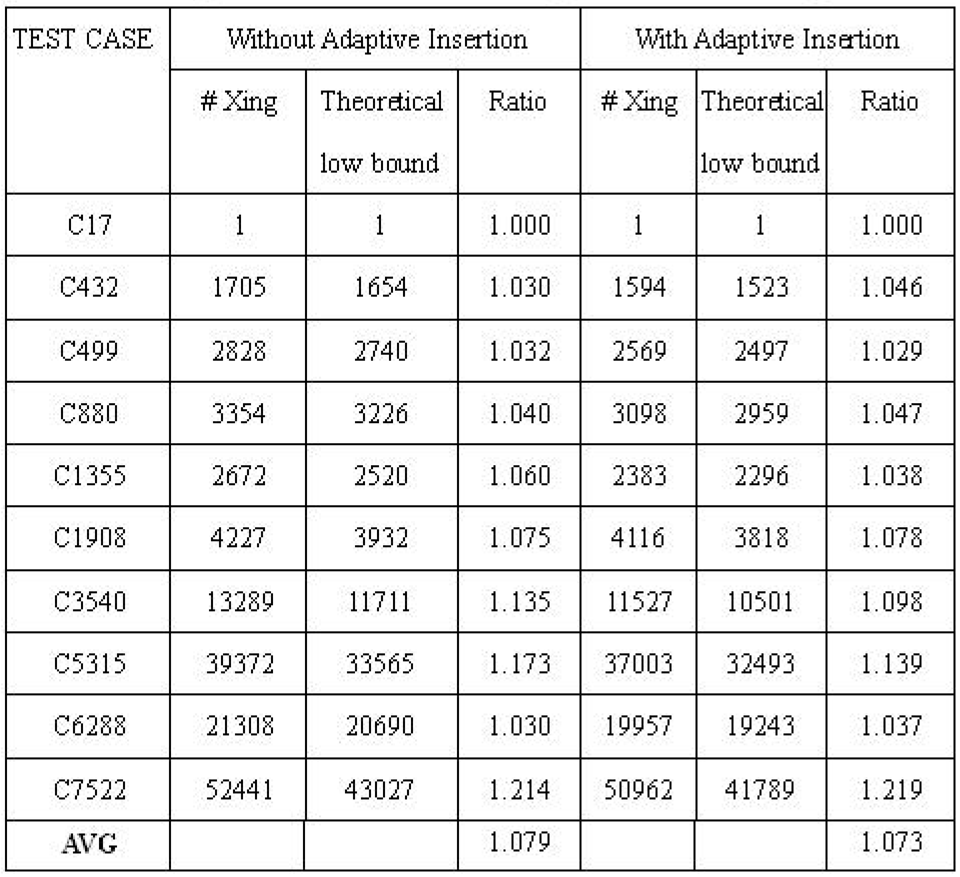
	Normalized with Barycenter		Normalized with Median	
	Without AI	With A I	Without AI	With A I
C499	0.77	0.62	0.62	0.50
C880	0.88	0.78	0.85	0.75
C1355	0.65	0.53	0.55	0.46
C1908	1.16	0.97	1.04	0.86
C3540	0.83	0.70	0.74	0.63
C5315	0.69	0.63	0.60	0.54
C6288	0.74	0.58	0.53	0.41
C7522	0.91	0.83	0.92	0.83
AVG	0.83	0.70	0.73	0.62

In our physical design flow, pseudo routing stage would check unroutable net existed or not and expand the height of the level. This is harmful to the area size of QCA layout. Fortunately, those unroutable net are rarely seen in the circuit. In our experiment, there were at most 2 expanded rows in a circuit.

In table 5, we compare our channel routing result based on our placement method with or with adaptive insertion to the theoretical lower bound. The theoretical lower bound of wire crossing for each circuit is equal to the sum of the minimum values of wire crossing for each pair of nets. Our channel routing heuristic result in average wire crossing only 7 ~ 8% above the theoretical lower bound.

Table5 . Comparison of wire crossing value to the theoretical lower bounds.

# Xing denotes the total wire crossing after channel routing



TEST CASE	Without Adaptive Insertion			With Adaptive Insertion		
	# Xing	Theoretical low bound	Ratio	# Xing	Theoretical low bound	Ratio
C17	1	1	1.000	1	1	1.000
C432	1705	1654	1.030	1594	1523	1.046
C499	2828	2740	1.032	2569	2497	1.029
C880	3354	3226	1.040	3098	2959	1.047
C1355	2672	2520	1.060	2383	2296	1.038
C1908	4227	3932	1.075	4116	3818	1.078
C3540	13289	11711	1.135	11527	10501	1.098
C5315	39372	33565	1.173	37003	32493	1.139
C6288	21308	20690	1.030	19957	19243	1.037
C7522	52441	43027	1.214	50962	41789	1.219
AVG			1.079			1.073

Figure 5-1 shows our synthesis result of C17 circuit, and Figure 5-3 shows its simulation result. We simulate our C17 circuit with 96000 samples, converge tolerance was set to 0.0001; radius of effect was set to 40.0 nm and at moat 1000 samples per iteration is allowed. In our synthesizer, C17 circuit is divided into 10 clocking zone, i.e. $2\frac{1}{2}$ QCA clock period. Thus there were $2\frac{1}{2}$ volatile output values before simulation output of the first input pattern. We mark those volatile output values by red rectangle in Figure 5-3.

To verify our simulation result, we simulate C17 circuit with the same input patterns with Modelsim and the result was shown in Figure 5-2. There were no different output values comparing of Figure 5-2 and Figure 5-3. In other word, our C17 circuit was correctly synthesized.

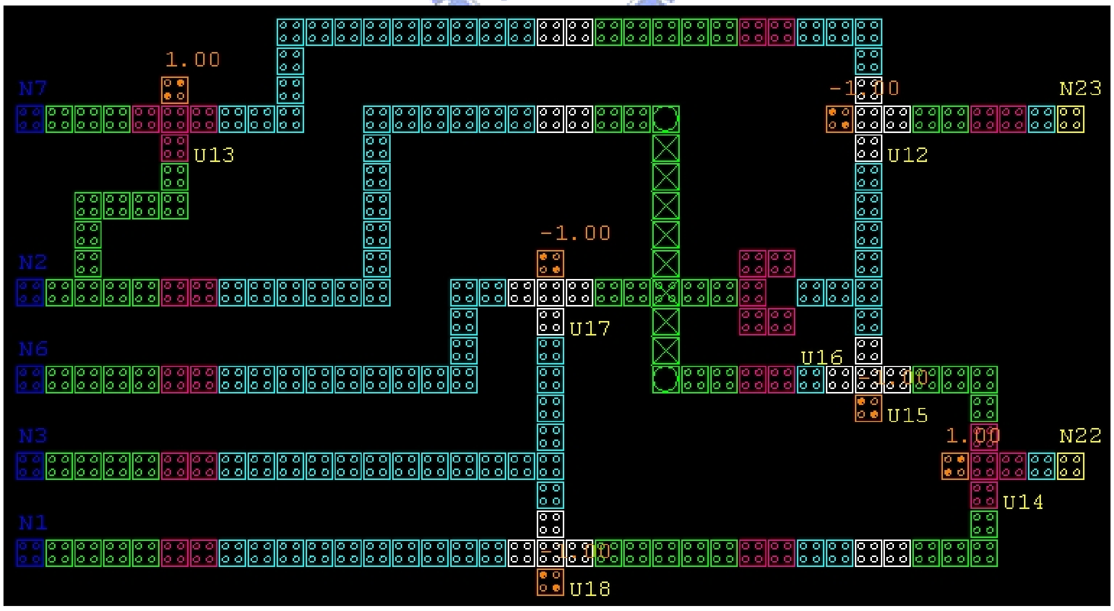


Figure 5-1 C17 circuit

Besides C17 circuit, we also verified C432 circuit with 100 random selected patterns. Figure 5-4 shows the simulation result of QCA Designer and Figure 5-5 shows the simulation result of Modelsim. Differing form simulating C17 circuit, we simulated each input pattern 25000 samples to obtain the correct simulation result. In C432 circuit, there were 18 volatile output values because there were 72 clocking zones in our C432 circuit. Due to the limitation of hardware resource, we didn't verify other synthesis circuit in ISCAS85.

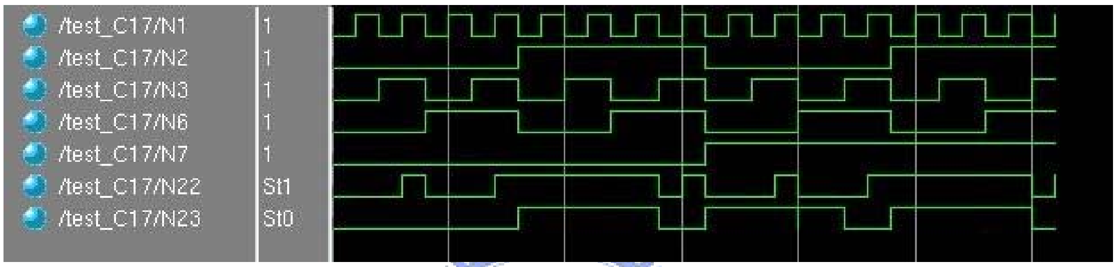


Figure 5-2 Simulation result of C17 by Modelsim

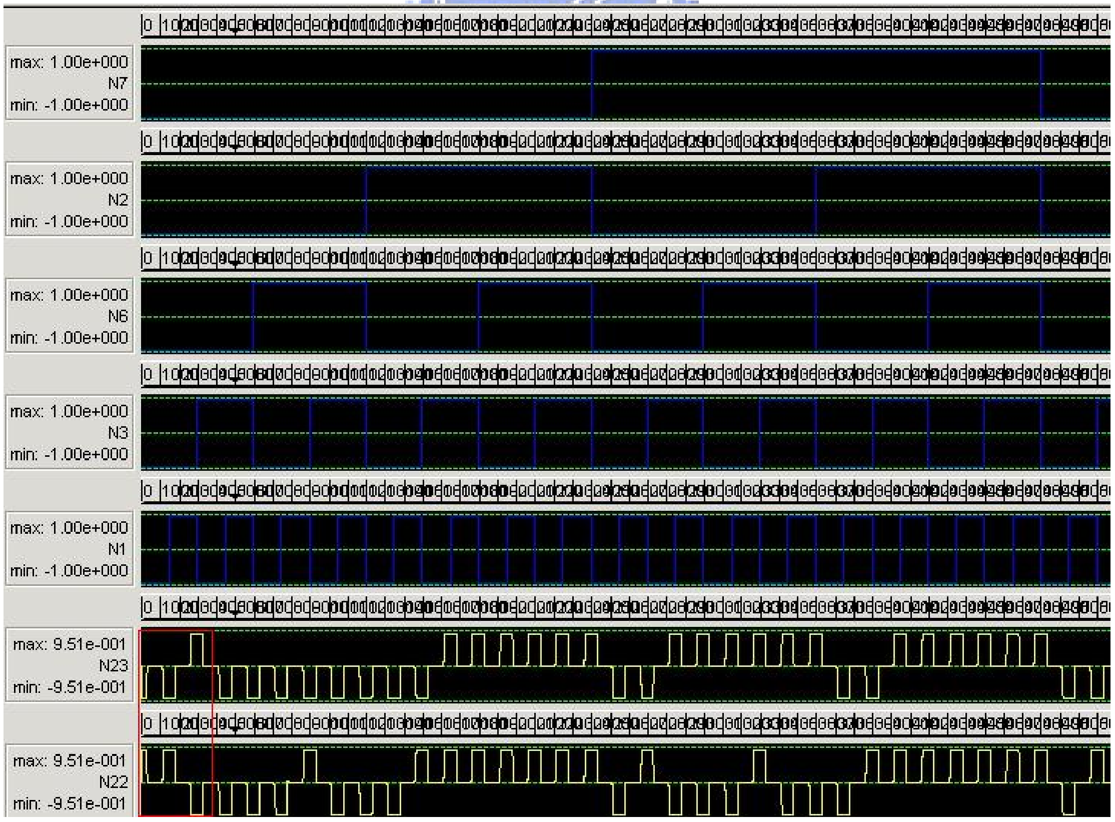


Figure 5-3 Simulation result of C17 by QCA Designer

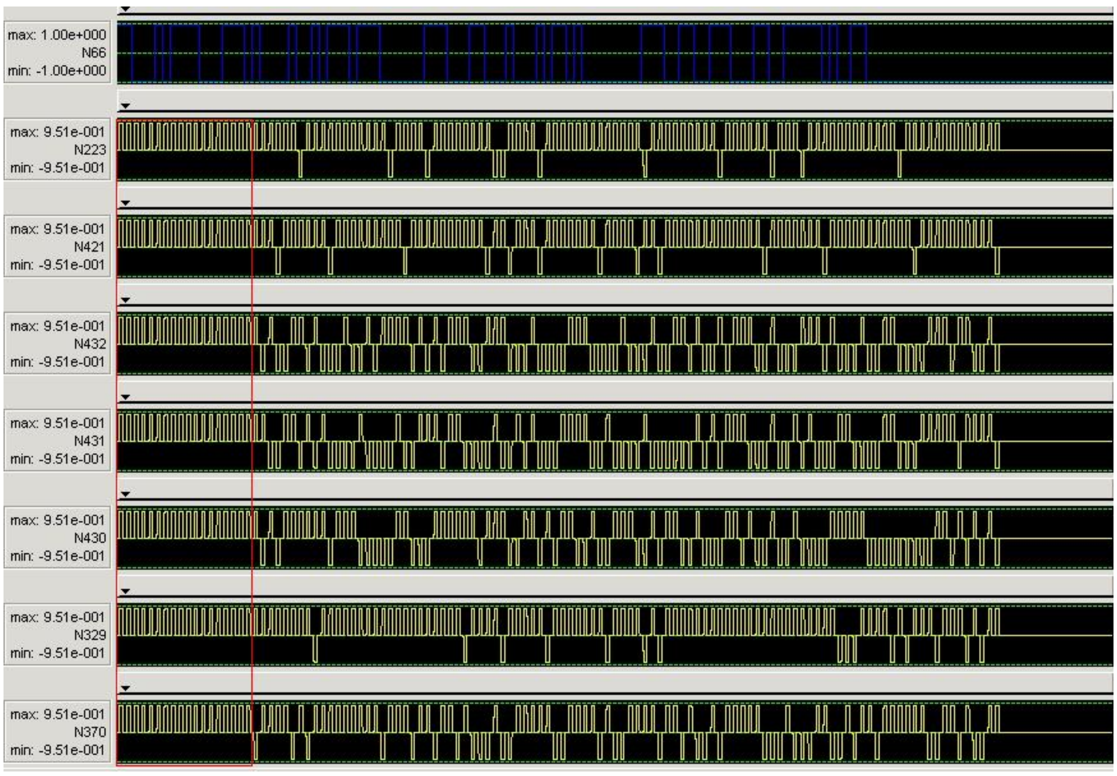


Figure 5-4 Simulation result of C432 by QCA Designer

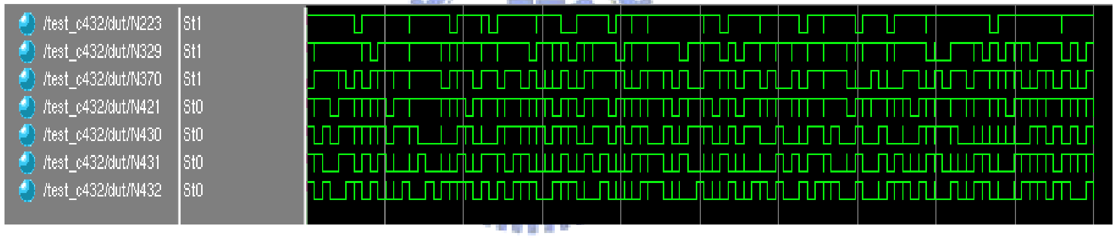


Figure 5-5 Simulation result of C432 by Modelsim

Chapter 6
Conclusions

In this paper we describe a physical design flow of QCA layout synthesis and wish to help generating synthesizable QCA circuits. From the experimental test, we found out wire crossing were strongly influenced the correctness of QCA circuit functionality. For this reason, we focus on generating QCA layout with minimum wire crossings. QCA layout of ISCAS benchmark circuit C17 and C432 were synthesized and verified functionality successfully.

Bibliography

[1] Craig S.Lent, P. Douglas Tougaw, and Wolfgang Porod. “Quantum Cellular Automata.” *Nanotechnology* 4, 49 , 1993.

[2] Michael T. Niemier. “Designing Digital Systems in Quantum Cellular Automata.” *MS CSE Thesis*, University of Notre Dame, 2000.

[3] Michael T. Niemier and Peter M. Kogge. “Exploring and Exploiting Wire-Level Pipelining in Emerging Technologies.” *ISCA*, 2001.

[4] Islamshah Amlani, Alexei O.Orlov, Geza Toth, Gary H. Bernstein, Craig S. Lent, Gregory L. Snider. “Digital Logic Gate Using Quantum-Dot Cellular Automata”. *Science Vol* 284 pp. 289 – 29, 1999.

[5] Gary H. Bernstein, “Quantum-dot cellular automata: computing by field polarization”, *Proceedings of the 40th conference on Design automation*, 2003.

[6] A.O.Orlov I. Amlani, G.H. Bernstein, C.S. Lent, and G.L. Snider, “Realization of a Functional Cell for Quantum-Dot Cellular Automata”, *Science* Vol. 277, pp. 928, 1997.

[7] G. L. Snider , A.O. Orlov, I. Amlani, X. Zuo, G. H. Bernstein, C. S. Lent, J. L. Merz, W. Porod, “Quantum-dot cellular automata: Review and recent experiments” , *Journal of Applied Physics* , Vol. 85, No. 8, pp. 4283-4285 , 1999.

[8] Tougaw, P. Douglas; Lent, Craig S.” Logical devices implemented using quantum cellular automata”. *Journal: Journal of Applied Physics,* Volume75, Issue 3, pp.1818-1825, 1994.

[9] D. Berzon, T.Fountain , “Computer Memory Structures using QCAs”, *Report No.98/1*.

[10] Dominic A. Antonelli, Danny Z. Chen, Timothy J. Dysart, Xiaobo S. Hu, Andrew B. Kahng, Peter M. Kogge, Richard C. Murphy, Michael T. Niemier “New ideas in placement: Quantum-Dot Cellular Automata (QCA) circuit partitioning: problem modeling and solutions” *Proceedings of the 41st annual conference on Design automation* , 2004.

[11] Yih Lang Li and Cheng Wen Wu, “Cellular Automata for Efficient Parallel Logic and Fault Simulation ," *IEEE Trans. Computer-Aided Design*, Vol. 14, no. 6, pp. 740-749, 1995.

[12] J. Nguyen, R. Ravichandran, S.K. Lim, M. Niemier , “Global placement for quantum-dot cellular automata based circuits” *Technical Report* GIT-CERCS-03-20, Georgia Institute of Technology ,2003.

[13] Ramprasad Ravichandran, Sung Kyu Lim, Mike Niemier , “Automatic cell placement for quantum-dot cellular automata” *INTEGRATION* , the VLSI journal , pp. 541-548, 2005.

[14] Brian Stephen Smith, Sung Kyu Lim “QCA channel routing with wire crossing minimization.” *ACM Great Lakes Symposium on VLSI*, pp. 217-220 , 2005.

[15] Ramprasad Ravichandran, Mike Niemier, “Eliminating Wire Crossings for Molecular Quantum-dot Cellular Automata Implementation.” ICCAD 2005

[16] Eade, P. and Whiteside S. “Drawing graphs in two layers.” *Theoretical Computer Science 131*, pp.361-374, 1994

[17] Matusewski, C. Schonfeld , R., and Molitor, P. “Using sifting for k-layer straightline crossing minimization” *Proc. 7$_{th}$ Graph Drawing Conference* , Number 1731 in LNCS , pp.217-224, 1999.

[18] Schonfeld, R., Gunter, W., Becker, B., and Molitor, P. “K-layer straightline crossing minimization by soeeding up shifting.” *Proc. 8$_{th}$ Graph Drawing Conference*, Number 1984 in LNCS, pp.253-258, 2000.

[19] Matthias Stallmann, Franc Brglez, Debabrata Ghosh, “Heuristics, Experimental Subjects, and Treatment Evaluation in Bigraph Crossing Minimization”, *Journal of Experimental Algorithmics* , 2001.

[20] Kernighan, B. W. and Lin, S. “An efficient heuristic procedure for partitioning graphs.” *Bell System Technical Journal*, pp. 291-307, 1970.

[21] Walus, K.; Dysart, T.J.; Jullien, G.A.; Budiman, R.A. “QCADesigner: a rapid design and Simulation tool for quantum-dot cellular automata”, *Nanotechnology, IEEE Transactions* on Volume 3, Issue 1, pp.26 – 31 , 2004.

[22] G. Toth, "Correlation and coherence in quantum-dot cellular automata", *Ph.D. Thesis*, University of Notre Dame, pp. 56-63, 2000.

[23] A. Gin, P. D. Tougaw, and S. Williams. “An alternative geometry for quantum-dot cellular automata”. *J. Appl. Phys.*, 85(12):8281–8286, 1999.