Available online at www.sciencedirect.com
SCIENCE DIRECT®
INTEGRATION, the VLSI journal 38 (2005) 541–548
INTEGRATION theVLSI journal
www.elsevier.com/locate/vlsi

# Automatic cell placement for quantum-dot cellular automata ☆

Ramprasad Ravichandran[a], Sung Kyu Lim[b,*], Mike Niemier[a]

[a]College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA
[b]School of Electrical and Computer Engineering, Georgia Institute of Technology, 777 Atlantic Drive NW, Atlanta 30305, GA 30332, USA

Received 14 July 2004; accepted 21 July 2004

**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 chemical molecules. In this paper we develop the first cell-level placement of QCA circuits under buildability constraints. We formulate the QCA cell placement as a unidirectional geometric embedding of k-layered bipartite graphs. We then present an analytical and a stochastic solution for minimizing the wire crossings and wire length in these placement solutions.


*MSC:* 94C15; 68W35; 03G12

*Keywords:* Quantum-dot Cellular Automata; Placement

## 1. Introduction

One approach to computing at the nano-scale is the quantum-dot cellular automata (QCA) [1,2] concept that represents information in a binary fashion, but replaces a current switch with a cell having a bi-stable charge configuration. A wealth of experiments have been conducted with

☆A short version (Ravichandran et al., 2004) is published in the Proceedings of ACM Great Lake Symposium on VLSI, 2004.

*Corresponding author. Tel.: 4048940373; fax: 4043851746.
*E-mail address:* limsk@ece.gatech.edu (S.K. Lim).

0167-9260/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2004.07.002

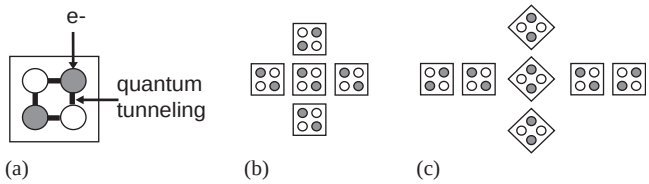



Fig. 1. Illustration of QCA device, majority gate, and wires.

metal-dot QCA, with individual devices, logic gates, wires, latches and clocked devices, all having been realized. In this article, we develop the first cell-level placement of QCA circuits. We formulate the QCA cell placement as a unidirectional geometric embedding of k-layered bipartite graphs. We then present an analytical and a stochastic solution for minimizing the wire crossings and wire length in these placement solutions. Our goal is to identify several objectives and constraints that enhance the buildability of QCA circuits and use them in our placement optimization process. The results are intended to define what is computationally interesting and could actually be built within a set of predefined placement constraints.

A QCA cell is illustrated in Fig. 1(a). Two mobile electrons are loaded into this cell and can move to different quantum dots by means of electron tunneling. Coulombic repulsion will cause the electrons to occupy only the corners of the QCA cell, resulting in two specific polarizations. The fundamental QCA logical gate is the three-input majority gate. It consists of five cells and implements the logical equation $AB + BC + AC$ as shown in Fig. 1(b). The QCA wire is a horizontal row of QCA cells and a binary signal propagates from left-to-right because of electrostatic interactions between adjacent cells as shown in Fig. 1(c). A QCA wire can also be comprised of cells rotated 45°. Here, as a binary signal propagates down the length of the wire, it alternates between a binary 1 and a binary 0 polarization. QCA wires are able to cross in the plane without the destruction of the value being transmitted on either wire as shown in Fig. 1(c).

Our work focus on the following undesirable design schematic characteristics associated with a near-to-midterm buildability point: large amounts of deterministic device placement, long wires, clock skew, and wire crossings. We will use CAD to: (1) identify logic gates and blocks that can be duplicated to reduce wire crossings; (2) rearrange logic gates and nodes to reduce wire crossings; (3) create shorter routing paths to logical gates (to reduce the risk of clock skew and susceptibility to defects and errors); and (4) reduce the area of a circuit (making it easier to physically build). Some of these problems have been individually considered in existing work for silicon-based VLSI design, but in combination, form a set of constraints unique to QCA requiring a unique toolset to solve them.

## 2. Problem formulation

QCA placement is divided into three steps: zone partitioning, zone placement, and cell placement. An illustration is shown in Fig. 2. The purpose of zone partitioning is to decompose an input circuit such that a single potential modulates the inner-dot barriers in all of the QCA cells that are grouped within a clocking zone. The zone placement step takes as input a set of

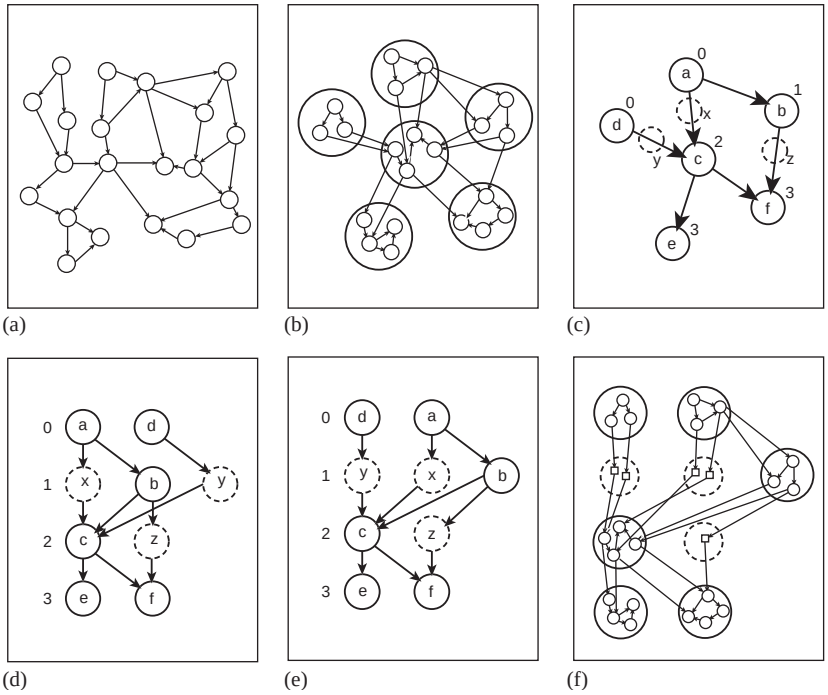



Fig. 2. Illustration of QCA placement steps. (a) input circuit represented with a DAG (directed acyclic graph), (b) zone partitioning, (c) wire block insertion, (d) zone placement, (e) wire crossing minimization at zone-level, (f) cell placement.

zones—with each zone assigned a clocking label obtained from zone partitioning. The output of zone placement is the best possible layout for arranging the zones on a two dimensional chip area. Finally, cell placement visits each zone to determine the location of each individual logic QCA cell—a cell used to build majority gates. Our recent work on zone partitioning and zone placement work is available in [3]. The focus of this article is on cell placement that is formally defined as follows:

**Definition 1.** Cell placement: we seek a placement of individual logic gates in the logic block so that area, wire crossing and wirelength are minimized. The following set of constraints exists during QCA cell placement: (1) the timing constraint: the signal propagation delay from the beginning of a zone to the end of a zone should be less than a clock period established from zone partitioning; (2) the terminal constraint: the I/O terminals are located on the top and bottom boundaries of each logic block; (3) the signal direction constraint: the signal flow among the logic QCA cells needs to be unidirectional–from the input to the output boundary for each zone.

The signal direction is caused by QCA's clocking scheme, where an electric field $E$ created by underlying CMOS wire is propagating in uni-directionally within each block. Thus, cell placement needs to be done in such a way to propagate the logic outputs in the same direction as $E$. In order to balance the length of intra-zone wires, we construct *cell-level k-layered bipartite graph* for each zone and place this graph. We define the *k-layered bipartite graph* as follows:

**Definition 2.** K-layered bipartite graph: a directed graph $G(V, E)$ is k-layered bipartite graph if (i) $V$ is divided into $k$ disjoint partitions, (ii) each partition $p$ is assigned a level, denoted $lev(p)$, and (iii) for every edge $e = (x, y)$, $lev(y) = lev(x) + 1$.

## 3. Cell placement algorithm

This section presents our cell placement algorithm, which consists of feed-through insertion, row folding, and wire crossing and wirelength optimization steps.

### 3.1. Feed-through insertion

In order to satisfy the relative ordering and to satisfy the signal direction constraint, the original graph $G(V, E)$ is mapped into a k-layered bipartite graph $G'(V', E')$ which is obtained by insertion of feed-through gates, where $V'$ is the union of the original vertex set $V$ and the set of feed-through gates, and $E'$ is the corresponding edge set. The following algorithm performs feed-through insertion.

```
feed-through_insertion(G(V, E))
if (V is empty) return;
n = V.pop();
if (n has no child with bigger level) return;
g = new feed-through;
lev(g) = lev(n) + 1;
for (each child c of n)
   g = parent(c); c = child(g);
   n = parent(g); g = child(n);
add g into G;
feed-through_insertion(G(V,E));
```

In this algorithm, we traverse through every vertex in the graph. For a given vertex, if any of the outgoing edges terminate at a vertex with topological order more than one level apart, a new feed-through vertex is added to the vertex set. The parent of the feed-through is set to the current vertex, and all children of the current vertex which have a topological order difference of more than one is set as the children of the feed-through. We do not need to specifically worry about the exact level difference between the feed-through and the child nodes, since this feed-through insertion is a recursive process. This algorithm runs in $O(k|V'|)$, where $k$ is the maximum degree of $V'$. Fig. 3 shows the graph before and after feed-through insertion.

### 3.2. Row-folding algorithm

After the feed-through insertion stage, some rows may have more gates than the average number of gates per row. The row with the largest number of gates defines the width of the entire zone, and hence the width of the global column that the zone belongs to. This increases the circuit

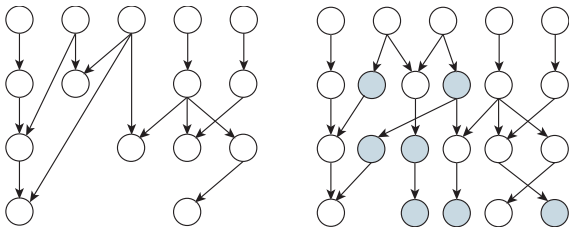


Fig. 3. Illustration of feed-through insertion, where a cell-level k-layered bipartite-graph is formed via feed-through nodes.

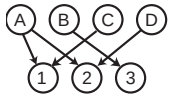
area by a huge factor. Hence, rows with a large number of cells are folded into two or more rows. This is done by inserting feed-through gates in place of the logic gates and moving the gates to the next row. Row-folding decreases the width of the row since feed-throughs have a lower width than the gate it replaces. A gate $g$ is moved into the next existing row if it belongs to the row that needs to be folded and all paths that $g$ belongs to contain at least one feed-through with a higher topological order than $g$. The reason for the feed-through condition is that $g$, along with all gates between $g$ and the feed-through can be pushed to a higher row, and the feed-through can be deleted without violating the topological ordering constraint. The following algorithm performs row folding.

```
row_folding(G, w)
if (w is a feed-through)
   return(TRUE);
if (w.level = G.max_level)
   return(FALSE);
RETVAL = TRUE;
k = w.outdegree;
i = 0;
while (RETVAL and i<k)
   RETVAL = row_folding(G, w.CHILD(i));
   i = i + 1;
return(RETVAL);
```

This algorithm returns true if a node can be moved, and false if a new row has to be inserted. If this feed-through criterion is not met, and the row containing $g$ has to be folded, then a new row is inserted and $g$ is moved into that row.

### 3.3. Wirelength and wire crossing minimization

A width-balanced k-layered bipartite graph is formed via feed-through insertion and row folding stages. This graph is placed in such a way that all cells of the same longest path length are placed in the same row. The next step is then to rearrange the cells in each row to reduce wire crossing. Wire crossing minimization is already NP-hard for bipartite graphs with two rows




(a)

| | 1 | 2 | 3 |
|---|---|---|---|
| A | 1 | 1 | 0 |
| B | 0 | 0 | 1 |
| C | 1 | 0 | 0 |
| D | 0 | 1 | 0 |

(b)

| | 1 | 2 | 3 |
|---|---|---|---|
| A | 1 | 2 | 2 |
| B | 0 | 0 | 1 |
| C | 1 | 1 | 1 |
| D | 0 | 1 | 1 |

(c)

| | 1 | 2 | 3 |
|---|---|---|---|
| A | 2 | 4 | 5 |
| B | 1 | 2 | 3 |
| C | 1 | 2 | 2 |
| D | 0 | 1 | 1 |

(d)

Fig. 4. Illustration of incremental wire crossing computation. (a) a bipartite graph with 3 wire crossings, (b) adjacency matrix of (a), (c) row-wise sum of (b) from left to right, (d) column-wise sum of (c) from bottom to top. Each entry in (d) now represent the total sum of entries in low-left sub-matrix. Using (b) and (d), wire crossing is $A2 \times B1 + B3 \times C2 = 3$, where $A2$ and $B3$ are from (b) and $B1$ and $C2$ from (d).

only [4]. Our approach for wire crossing minimization in $k$-layered bipartite graphs is to use a well-known barycenter heuristic [4] to build the initial solution and refine it with Simulated Annealing. In barycenter heuristic, the nodes in the top layer are fixed and used to rearrange the nodes in the bottom layer. For each node $v$ in the bottom layer, we compute the center of mass, i.e., $m(v) = \sum_{u \in FI(v)} column(u)/|FI(v)|$, where $FI(v)$ denotes the fan-in nodes of $v$. These nodes are then sorted in an increasing order of $m(v)$ and placed from the left-most column. During Simulated Annealing, a move is performed by swapping two randomly chosen gates in the same row in order to minimize the total wirelength and wire crossing. We initially compute the wirelength and wire crossing and incrementally update these values after each move so that the update can be done much faster. This speedup allows us to explore a greater number of candidate solutions, and as a result, obtain better quality solutions.

We use the adjacency matrix to compute the number of wire crossings. In a bipartite graph, there is a wire crossing between two layers $v$ and $u$ if $v_i$ talks to $u_j$ and $v_x$ talks to $u_y$, where $i$, $j$, $x$, and $y$ denote the relative positional ordering of the nodes, and either, $i<x<j<y$ or $i<x<y<j$ or $x<i<y<j$ or $x<i<j<y$ without loss of generality. In terms of an adjacency matrix, this can be regarded as if either the point $(i,j)$ is included in the lower left sub-matrix of $(x,y)$ or vice versa. Fig. 4 shows an example of wire crossing computation. The total crossing is computed by adding the product of every matrix element and the sum of its left lower sub-matrix entries. i.e. $\sum(A_{ij} \times \sum\sum A_{xy})$, where $i+1<x<n$ and $1<y<j-1$. However, this method is computationally expensive if we have to perform it frequently. In our incremental wire crossing calculation, we first take the row-wise sum of all entries as in Fig. 4(c). Then we use this to compute the column-wise sum as in Fig. 4(d). Finally, we multiply all the entries in the original matrix and the column-wise sum matrix to compute the total wire crossing–each entry $(r,c)$ in the original matrix is multiplied by the entry $(r+1, c-1)$ in the column-wise sum matrix. In the Simulated Annealing process, when we swap two nodes, it is identical to swapping the corresponding rows in the above matrices. Hence, it is enough if we just update the values of the rows in between the two rows that are being swapped.

## 4. Experimental results

Our algorithms were implemented in C++/STL, compiled with gcc v2.96 run on Pentium III 746 MHz machine. The benchmark set consists of seven biggest circuits from ISCAS89 and five

Table 1
QCA cell placement results

| | Analytical | | SA + WL | | SA + WC | | SA + WL + WC | |
|---|---|---|---|---|---|---|---|---|
| | wire | xing | wire | xing | wire | xing | wire | xing |
| b14 | 5586 | 1238 | 28680 | 23430 | 54510 | 3740 | 5113 | 4948 |
| b15 | 9571 | 1667 | 23580 | 40400 | 69030 | 7420 | 8017 | 8947 |
| s13207 | 3119 | 548 | 14060 | 15530 | 30610 | 1450 | 3250 | 1982 |
| s15850 | 3507 | 634 | 18610 | 22130 | 42700 | 2140 | 3919 | 2978 |
| s38417 | 9414 | 1195 | 45830 | 48400 | 80240 | 7320 | 9819 | 9929 |
| s38584 | 19582 | 4017 | 59220 | 75590 | 140130 | 9820 | 20101 | 33122 |
| s5378 | 1199 | 156 | 6280 | 6690 | 13600 | 730 | 1344 | 841 |
| s9234 | 2170 | 205 | 10720 | 11540 | 23290 | 980 | 1640 | 2159 |
| Ave | 4192 | 741 | 16980 | 19950 | 38950 | 2740 | 3880 | 6878 |
| Ratio | 1.00 | 1.00 | 4.05 | 26.9 | 9.29 | 3.69 | 0.92 | 9.27 |
| Time | 180 | | 604 | | 11280 | | 12901 | |

biggest circuits from ITC99 suites due to the availability of signal flow information. Table 1 shows our cell placement results where we report net wirelength and number of wire crossings for the circuits using our analytical solution and all three flavors of our Simulated Annealing algorithm. We observe in general that analytical solution is better than all three flavors of the Simulated Annealing methods, except the wirelength of SA + WL + WC. But, the tradeoff in wire crossings makes the analytical solution more viable, since wire crossings pose a bigger barrier than wirelength in QCA architecture. One interesting note is that when comparing amongst the three flavors of Simulated Annealing we find that SA + WC has the best wire crossing number. But surprisingly, in terms of wirelength, SA + WL does not outperform SA + WL + WC. We speculate that this behavior is because lower number of wire crossings has a strong influence on wirelength, but smaller wirelength does not necessarily imply smaller crossing.

## 5. Conclusions and ongoing works

In this article, we proposed a QCA cell placement problem and present an algorithm that will help automate the process of design within the constraints imposed by physical scientists. Work to address QCA routing and node duplication for wire crossing minimization are underway. The outputs from this work and the work discussed here will be used to generate computationally interesting and optimized designs for experiments by QCA physical scientists.

## References

[1] R. Ravichandran, N. Ladiwala, J. Nguyen, M. Niemier, S.K. Lim, Automatic cell placement for quantum-dot cellular automata, in: Proceedings of the Great Lakes Symposium on VLSI, 2004.

[2] I. Amlani, A. Orlov, G. Toth, G. Bernstein, C. Lent, G. Snider, Digital logic gate using quantum-dot cellular automata, Science (1999) 289–291.
[3] 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.
[4] K. Sugiyama, S. Tagawa, M. Toda, Methods for visual understanding of hierarchical system structures, IEEE Trans. Syst. Man Cybern. (1981) 109–125.