

Automatic Placement for Quantum Cell Automata

Ramprasad Ravichandran[†], Nihal Ladiwala[‡], Jean Nguyen[‡], Sung Kyu Lim[‡], and Mike Niemier[†]

[†]College of Computing

[‡]School of Electrical and Computer Engineering

Georgia Institute of Technology

{*raam@cc, gte568t@prism, jnguyen@ece, limsk@ece, mniemier@cc*}.gatech.edu

Abstract: Quantum-dot Cellular Automata (QCA) is a novel computing mechanism that can represent binary information based on spatial distribution of electron charge configuration in chemical molecules. It has the potential to allow for circuits and systems with functional densities that are better than end of the roadmap CMOS, but also imposes new constraints on system designers. Several recent works concentrated on partitioning and global placement in the context of QCA schematics based on constraints imposed by this architecture. In this paper we develop the first cell-level placement of QCA circuits, where the given circuit is assumed to be partitioned into 4-phase asynchronous QCA timing zones. We formulate the QCA cell placement in each timing zone 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. Results provide designs of circuits and systems that will be used to develop computationally interesting designs for chemists who are currently preparing to build the patterns and substrates required for real QCA circuits.

1. INTRODUCTION

Nano technology and devices will have revolutionary impact on the CAD field. Similarly, CAD research at circuit, logic and architectural levels for nano devices can provide valuable feedbacks to nano research and illuminate ways for developing new nano devices. It is time for CAD researchers to play an active role in nano research. One approach to computing at the nano-scale is the quantum-dot cellular automata (QCA) concept that represents information in a binary fashion, but replaces a current switch with a cell having a bi-stable charge configuration. QCA devices can be realized in metal [2], or with chemical molecules [1]. A wealth of experiments have been conducted with metal-dot QCA, with individual devices [2,3], logic gates [3,4,5], wires [4], latches [3], and clocked devices [3,6], all having been realized. This advancement is followed by various recent efforts in developing CAD tools for QCA based circuits and systems [18,19].

In this paper we develop the first cell-level placement of QCA circuits, where the given circuit is assumed to be partitioned into 4-phase asynchronous QCA timing zones. We formulate the QCA cell placement in each timing zone

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. Results provide designs of circuits and systems that will be used to develop computationally interesting designs for chemists who are currently preparing to build the patterns and substrates required for real QCA circuits.

2. PRELIMINARIES

2.1. QCA Devices

“3-dot cells” are fundamental building blocks of QCA cells, which been researched and simulated for the last two decades. This molecule shown in Figure 1 forms a ‘v’-shape, and charge can be localized on any one of the three dots at the “points” of the ‘v’. If charge is localized on one of the top two dots, the cell will encode a binary 1 or 0. Whether or not charge is in the top two dots (the *active* state) or the lower dot (the *null* state) can be determined by an electric field that will raise or lower the potential of the central dot relative to the top two dots [1]. When considering basic cell-to-cell interactions, binary 1s and 0s are physically represented by the dipole moments of QCA molecules. Dipole moments are formed by the way that charge is localized within certain sites of a QCA molecule and how that charge can tunnel between these sites [7]. In the presence of a strong driver dipole, a larger amount of energy is required to excite a cell into a mistake state [8].

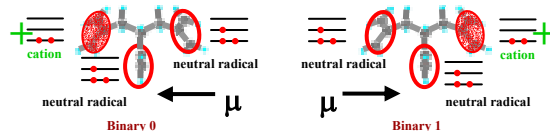


Figure 1. Schematic representation of molecular QCA

Circuit elements are shown and described here in terms of 4-dot QCA cells. A 4-dot cell could simply be formed by two adjacent 3-dot but are also being engineered as explicit molecules [9]. 4-dot cells are ideal because of symmetry. Binary information is stored and moved with quadrupole moments. The fundamental QCA logical gate is a three-input majority gate (Figure 2a). Computation is performed by driving the *device* cell to its lowest energy state, which will occur when it assumes the polarization of the majority

of three inputs. Data can be moved in a QCA circuit with a row of QCA cells. A binary signal propagates from left-to-right because of electrostatic interactions between adjacent cells. A QCA “wire” can also be comprised of cells rotated 45-degrees (Figure 2c). Here, a binary signal will alternate between a binary 1 and a binary 0. The majority voting function can be reduced to an AND or OR function by setting an input to a 0 or a 1. Inversion is also possible and QCA’s logic set is thus functionally complete. Finally, QCA wires of different orientations possess the unique property that they are able to cross in the plane without destroying the value being transmitted on either wire (Figure 2c). This is most important as, at present, all layout is assumed to be two-dimensional.

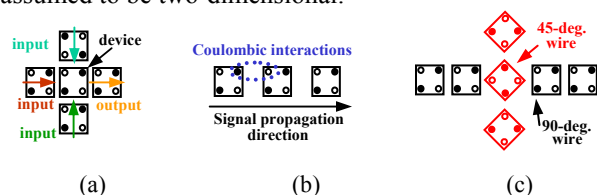


Figure 2. (a) Majority gate, (b) 90° wire, (c) wire crossing.

A clocking mechanism that will allow for a QCA device to transition from a monostable, null state to a bistable, active state, and then back to a monostable, null state is also desired for molecular QCA cells. QCA’s clock is typically viewed to be an electric field that controls barriers within a QCA cell, and in turn controls whether or not excess charge in a QCA cell can represent a binary 1 or 0. It was first characterized as having 4 phases. In the *switch* phase, QCA cells begin un-polarized with inter-dot potential barriers low. Then, barriers are raised, QCA cells become polarized according to the state of their driver, and computation occurs. Raised barriers suppress electron tunneling and cell states are fixed. In the *hold* phase, barriers remain high so outputs of a subarray can be used as inputs to another subarray. In the *release* phase, barriers are lowered and cells relax to an unpolarized state. In the *relax* phase, barriers stay lowered and cells remain unpolarized [10].

A physical array of QCA cells can be divided into *zones* that offer the advantage of multi-phase clocking and group pipelining. For each zone, a single potential would modulate the inter-dot barriers in all of the cells in a given zone. This would allow one zone of QCA cells to perform a calculation, have its state frozen by the raising of inter-dot barriers, and have the output of that zone act as the input to a successor zone. For molecular QCA, the four phases of a clock signal could take the form of time-varying, repetitious voltages applied to silicon wires embedded underneath a substrate to which QCA cells were attached. Every fourth wire would simultaneously receive the same voltage and neighboring wires see delayed forms of the same signal [11]. The charge and discharge of the clocking wires will move the area of activity across the molecular layer of QCA cells. Computation occurs at the “leading edge” of the

applied electric field. Computation would move across the circuit in a continuous “wave” [3,10].

2.2. Motivation for QCA CAD Research

One might argue that it would be premature to perform any systems-level study of an emergent device while the physical characteristics of a device continue to evolve. However, it is important to note that many emergent, nano-scale devices are targeted for computational *systems* – and to date, most system-level studies have been proposed by physical scientists, and usually end with a demonstration of a functionally complete logic set or a simple adder. Useful and efficient computation will involve much more than this, and, in general, it is important to provide scientists with a better idea of how their devices should function. This coupling can only lead to an accelerated development of functional and interesting systems at the nano-scale. More specifically, with QCA, chemists are currently preparing to test the self-assembly process and its building blocks described in Section 2. Experiments could begin within one year and initial work will study the selective attachment and patterning of molecules to DNA tiles [12]. A significant and desired end result of this work would be the process of patterning itself. (In fact, initial experiments will not even use QCA molecules, but rather molecules that can be analyzed with Forster Resonance Enhanced Transfer of fluorescence as applied to DNA or RNA structure determination problems [12]). Thus, *our* work can provide the chemists with computationally interesting patterns – the *real* and eventual desired end result.

3. QCA CELL PLACEMENT

3.1. Problem Formulation

QCA placement is divided into three steps: *zone partitioning*, *zone placement*, and *cell placement*. 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. Unless QCA cells are grouped into zones to provide zone-level clock signals, each individual QCA cell will need to be clocked. The wiring required to clock each cell individually would easily overwhelm the simplicity won by the inherent local interconnectivity of a QCA architecture. However, because the delay of the biggest partition also determines the overall clock period, the size of each partition must also be determined carefully. In addition, four-phase clocking imposes a strict constraint on how to perform partitioning. The zone placement step takes as input a set of 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.

The input to the cell placement is zone placement result, where all logic/wire blocks at the same clocking level are placed in the same row. Then the output of cell placement is an arrangement of QCA cells in each logic block such that the wire length, wire crossing, and congestion are minimized while satisfying the timing, area, signal direction, terminal constraints as well as QCA specific design rules. The reconvergent path problem does not exist in cell placement—it is perfectly fine to have unbalanced reconvergent path lengths among the logic gates in each logic block. The reason is that correct output values will eventually be available at the output terminals in each block if the clock period is longer than the maximum path delay in each block. We determine the clock period based on the maximum path delay among all logic/wire blocks, so the reconvergent path problem does not exist anymore.

However, the following set of constraints exists during QCA cell placement: (i) timing constraint: signal propagation delay from the beginning to the end of the zone should be kept under the clock period computed from zone partitioning (maximum zone delay), (ii) area constraint: the placement area/dimension for each logic block is fixed, (iii) terminal constraint: the IO terminals are located on the top and bottom boundaries of each logic block, (iv) 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, and (v) design rules: we enforce various layout rules for QCA circuits including minimum/maximum cell/wire spacing and wire length, allowable cell off-centeredness and rotation, circuit densities, power dissipation, etc. The area and terminal constraints are inherited from zone partitioning and zone placement results. Each zone may have multiple inputs and multiple outputs, which requires that the topological ordering must match between the input and output of neighboring zones. 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 .

3.2 Construction of K-layer Bipartite Graphs

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 $G^2(V',E')$ 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. Figure 3 gives the pseudo-code for the recursive feed-through insertion algorithm. In this algorithm, we traverse through every vertex in the vertex set of 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 $O(KV')$, where K is the degree of the graph vertex v' of the graph G^2 . Figure 4a shows the graph before feed-through insertion and Figure 4b shows the graph after feed-through insertion. A trivial result of this stage is that all short paths have a set of feed-throughs between the last logical gate in the path and last row.

```

-----
INSERT-FT(G,V)
IF (V is not EMPTY)
  W = V.POP();
  K = W.OUTDEGREE;
  N = 0;
  INSERT = FALSE;
  WHILE (N < K)
    If (W.CHILD(N).LEVEL>W.LEVEL+1)
      INSERT = TRUE; BREAK;
    N = N+1;
  IF (INSERT)
    L = NEW GATE;
    L.SET_LEVEL(W.LEVEL + 1);
    L.SETPARENT(W);
    W.SETCHILD(L);
    G.ADDVERTEX(L);
    V.ADD(L);
    WHILE (N<K AND K>0)
      If (W.CHILD(N).LEVEL>W.LEVEL+1)
        W.CHILD(N).REMOVEPARENT(W);
        W.CHILD(N).ADDPARENT(L);
        L.ADDCHILD(W.CHILD(N));
        W.REMOVECHILD(W.CHILD(N));
        N = N-1;
        K = K-1;
    N = N+1;
  INSERT-FT(G,V);
-----

```

Figure 3: Feedthrough Insertion Algorithm

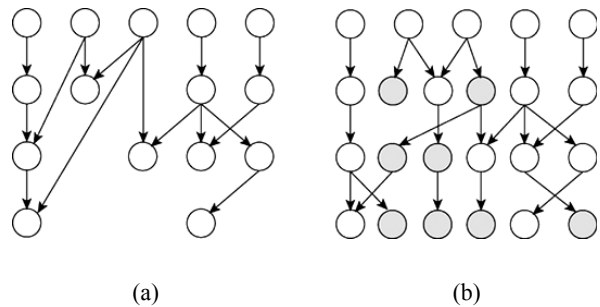


Figure 4. Before and after feed-through insertion. Shaded nodes indicate feed-throughs.

3.3 Row-folding Algorithm

After the feed-through insertion stage, some rows in G' may have much more number of 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 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, γ is moved into the next existing row if it belongs to the row that needs to be folded and all paths that γ belongs to contain at least one feed-through with a higher topological order than γ . The reason for the feed-through condition is that γ , along with all gates between γ and the feed-through can be pushed to a higher row, and the feed-through can be deleted without violating the topological ordering constraint. Figure 5 shows the pseudo-code for testing if a gate can be moved into an existing row. The 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 γ has to be folded, then a new row is inserted and γ is moved into that row.

```

-----
CHECK_FT(G,W)
IF(W_IS_A_FEEDTHROUGH)
    RETURN TRUE;
RETURN FALSE;
IF(W.LEVEL = G.MAX_LEVEL)
    RETURN FALSE;
RETURN TRUE;
K = W.OUTDEGREE;
I = 0;
WHILE(RETVAL & I<K)
    RETVAL = CHECK_FT(G,W.CHILD(I));
    I = I+1;
RETURN RETVAL;
-----

```

Figure 5. Row folding algorithm

The number of gates that need to be moved from a row that needs folding to a new row is given by the following trivial calculation. Let η be the number of gates that need to be moved to the next row. Let μ be the original number of gates in the row, and let M be the maximum number of gates allowed in a row. Further, let α be the ratio of the width of a feed-through to the width of the gate. Since width of a gate is always greater than the width of a feed-through, $\alpha < 1$. For every gate that is moved to a new row, a feed-through has to be inserted in its original place. Hence, after moving η to the next row, the width of the original row will now be $\mu - \eta + \alpha\eta = M$, so $\eta = (\mu - M)/(1 - \alpha)$. This calculation is repeated for the next row if η is itself greater than the constraint M . The principal reason for increasing the height of a zone rather than increasing the width of the

zone is that the width of global column that the zone belongs to is much smaller than height of the column since the aspect ratio of the entire circuit layout is close to unity.

3.4. Wire length and Wire Crossing Minimization

At the end of the row-folding algorithm, we have a legal QCA circuit. The next stage in the cell placement algorithm is to optimize this layout to minimize the number of wire crossings and net wire length. We investigated and compared an analytical solution with a stochastic solution. We used the *barycenter heuristic* [13] for the analytical solution and *simulated annealing* for the stochastic algorithm. The analytical method only considers wire crossings since there is a strong correlation between wire length and number of wire crossings [17].

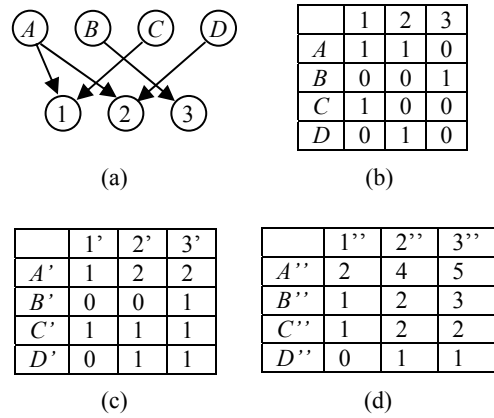


Figure 6. Illustration of wire crossing computation. (a) given graph, (b) initial adjacency matrix, (c) row-wise sum, (d) column-wise sum.

To compute the net wire length in a circuit we traverse through every vertex and accumulate the difference between the column numbers of the vertex and all of its children. This runs in $O(N)$, where N is the number of vertices. But, during the first calculation, we store the sum of all outgoing wire lengths in every vertex. This enables us to incrementally update if the position of only one node changes. A node cannot change its row number since at this stage the topological level is fixed. If a node changes its position within a level, then it is enough to calculate the difference in position with respect to its neighbors alone. Hence, subsequent wire length calculation is reduced to $O(K)$ where K is the node's vertex degree.

Wire crossing computation can be done with either the adjacency list or matrix, depending on the sparseness of the graph. We used the adjacency matrix to compute the number of wire crossings in a graph. In a 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 in 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 in the lower left sub-matrix of (x,y) or vice versa, there is a crosstalk. Hence, our solution is to count the number of such occurrences. If this counting is done unintelligently, it can be in the order of $O(n^4)$. Our algorithm to compute the number of wire crossings runs in $O(n^2)$.

Figure 6 shown and example of wire crossing computation. The graph in Figure 6a can be represented by the adjacency matrix shown in Figure 6b. The number of crossings in the diagram is 3. This can be obtained from the matrix by adding the product of every matrix element and the sum of its left lower matrix elements. i.e. the number of crossings = $\sum(A_{ij} \times \sum A_{xy})$, where $i+1 < x < n$ and $1 < y < j-1$. This formula gives a good intuition of the process but is computationally very expensive. We illustrate our method of calculating the same result. First we take the row-wise sum of all entries. Then we compute the column-wise sum. Finally, we multiply all the entries in the matrix with its lower-left neighbor's value and the sum of these products gives us the number of crossings. Then, we traverse through the original matrix and multiply every element with the element corresponding to its lower-left neighbor in the above matrix $O(n^2)$. i.e. $A1 \times (-) + A2 \times (B''1'') + B3 \times (C''2'') + C1 \times (-) + D2 \times (-) = 3$. In the simulated annealing process, when we swap two nodes in G'' , 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. The pseudo-code for this incremental algorithm is given in Figure 7.

```

-----
CALCXROWS (R1, R2, MATRIX)
IF (R2 < R1)
    RETURN CALCXROWS (R2, R1, MATRIX);
LET SUM = POS = NEG = DIFF = j = 0;
WHILE (J < NumRows)
    TEMP = DIFF;
    I = R2-1;
    WHILE (I > R1)
        SUM = SUM + MATRIX[I][j] * (POS-NEG);
        DIFF = DIFF + MATRIX[i][j];
        I = I + 1;
    SUM = SUM - MATRIX[R1][j] * (TEMP+NEG);
    SUM = SUM + MATRIX[R2][j] * (TEMP+POS);
    POS = POS + MATRIX[i][j];
    NEG = NEG + MATRIX[R2][j];
RETURN SUM;
-----

```

Figure 7. Incremental wire-crossing computation.

3.5. Optimization Engine

A widely used method for minimizing wire crossings in a graph (introduced by Sugiyama et al. [13] and Carpano [14]) is to map the graph into a k -layer bipartite graph. The vertices within a layer are then permuted to minimize wire crossings. This method maps well to this problem as we need to only consider the latter part of the problem (since

the clocking constraint yields us the k -layer bipartite graph). Still, even in a two-layer graph, minimizing wire-crossings is NP-hard [13]. Some of the common heuristics used to solve the one-sided crossing minimization are the barycenter heuristic [13], the split heuristic [14], the greedy-switch heuristic [15], median heuristic [16], stochastic heuristic [15], and the assign heuristic [14]. Amongst many heuristics proposed, the barycenter heuristic has been found to be the best heuristic in the general case for this class of problems. Therefore, an analytical wire crossing minimization method based on the barycenter algorithm was implemented.

In simulated annealing, a move is done by randomly choosing a level in the graph and then swapping two randomly chosen gates $[g_1, g_2]$ in that level in order to minimize the total wire length and wire crossing. In our implementation, the initial calculation of the wire length takes $O(n)$ and updating wire crossing takes $O(n^2)$ where n is the number of nodes in a layer of the bipartite graph. In our approach, we initially compute the wire length and wire crossing and incrementally update these values after each move so that the update can be done much faster as illustrated above. This speedup allows us to explore a greater number of candidate solutions, and as a result, obtain better quality solutions. We set the initial temperature such that roughly 50% of the bad moves were accepted. The final temperature was chosen such that less than 5% of the moves were accepted. We used three different cost functions. The first cost function only optimized based on the net wire length. The second cost function evaluated the number of wire crossings, while the last cost function looked at a weighted combination of both. The weights used were the ratio between the wirelength and the number of wire crossings obtained in the analytical solution.

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 six circuits from ISCAS89 and two circuits from ITC99 suites. We performed cell placement for these circuits based on QCA's structure and building blocks. There was an average of around 100 ± 10 gates per partition in each of the circuits. Table 1 shows our cell placement results where we report net wire length and number of wire crossings for the circuits using our analytical solution and all three flavors of our simulated annealing algorithm. We further tried simulated annealing from analytical start, and the results were identical to analytical solution. We observe in general that analytical solution is better than all three flavors of the Simulated Annealing methods, except in terms of wire length in the case of the weighted Simulated Annealing process. But, the tradeoff in wire crossings makes the analytical solution

Table 1. Cell placement results. We report wirelength (wl) and wire crossing (wc) for both analytical and Simulated Annealing based methods.

ckts	Analytical		SA+Wl		SA+Wc		SA+Wl+Wc	
	wl	wc	wl	wc	wl	wc	wl	wc
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
runtime	180		604		11280		12901	

more viable, since wire crossings pose a bigger barrier than wire length in QCA architecture.

One interesting note is that when comparing amongst the three flavors of simulated annealing we find that simulated annealing with wire crossing minimization alone has the best wire crossing number, but surprisingly, in terms of wire length, the simulated annealing procedure with wire length alone as the cost function is not as good as the simulated annealing procedure which optimizes both wire length and wire crossing. We speculate that this behavior is because lower number of wire crossings has a strong influence on wire length, but smaller wire length does not necessarily dictate lower number of crossings in our circuits.

5. CONCLUSIONS AND ONGOING WORKS

In this paper, we presented an algorithm that addressed the QCA cell placement problem. This will help automate the process of design within the constraints imposed by physical scientists. We are currently working on wire routing for QCA circuits. We are also studying logic duplication heuristics to completely remove all wire crossings. We hope this systems-level research can integrate with the physical device development to speed up the lab to market time of QCA circuits.

References

- [1] Lieberman M., Chellamma S., Varughese B., Wang Y., Lent C., Bernstein G.H., Snider G., and Peiris F.C. Quantum-dot cellular automata at a molecular scale. *Annals of the New York Academy of Science*, 960 (April 2002), 225-239.
- [2] Amlani I., Orlov A. O., Snider G., and Lent C.S.. Demonstration of a func. quantum-dot cellular automata cell. *J. Vac. Sci. Technol. B*, 16 (1998), 3795-3799.
- [3] Lent C.S., Snider G.L., Bernstein G., Porod W., Orlov A., Lieberman M., Fehlner T., Niemier T., and Kogge P. Quantum-Dot Cellular Automata. (2003).
- [4] Snider G.L., Orlov A.O., Amlani I., Bernstein G.H., Lent C.S., Merz J.L., and Porod W.. Quantum-dot cellular automata: Line and majority gate logic. *Jpn. J. of Applied Physics*, 38 (199) 7227-7229.
- [5] Amlani I., Orlov A., Toth G., Bernstein G., Lent C.S., and Snider G.L. Digital logic gate using quantum-dot cellular automata. *Science*, 284 (1999), 289-291.
- [6] Kummamuru R.V., Timler J., Toth G., Lent C.S., Ramasubramaniam R., Orlov A., and Bernstein G.H. Power gain in a quantum-dot cellular automata latch. *Applied Physics Letters*, 81(2002), 1332-1334.
- [7] Mathews C.K., van Holde K.E., and Ahren K.G. *Biochemistry*. Add. Wesley Longman, San Francisco, 2000.
- [8] Lent C.S., Isaksen B., and Lieberman M. Molecular Quantum-dot Cellular Automata. *J. Am. Chem. Soc.*, 125, (2003), 1056-1063.
- [9] LaBean T. H., Jens Kopatsch H.Y., Liu F., Winfree E., Reif J.H., and Seeman N.C. Construction, analysis, ligation, and self-assembly of dna triple crossover complexes. *J. Am. Chem. Soc.*, 122 (2000), 1848-1860.
- [10] Tougaw P.D. and Lent C.S. Logical devices implemented using quantum cellular automata. *Journal of Applied Physics*, 75 (1994), 1818.
- [11] Hennessy K. and Lent C.S. Clocking of molecular quantum-dot cellular automata. *Journal of Vacuum Science and Technology B*, 19,5(Sept-Oct 2001), 1752-1755.
- [12] Clegg R.M., Murchie A.I.H., Zechel A., and Lilley D.M.J.. Observing the helical geometry of double-stranded DNA in Solution by Fluorescence Resonance. *Proceedings of the National Academy of Sciences of the United States of America*, 90(7): 2994-2998, Apr 1, 1993.
- [13] Sugiyama K., Tagawa S., and Toda M. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Trans. Syst. Man., Cybern.*, SMC-11 (1981), 109-125.
- [14] Carpano M.J. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Trans. on Syst. Man., Cybern.*, 10,11, (1980), 705-715.
- [15] Eades P. and Wormald N. Edge crossings in drawing of bipartite graphs. *Algorithmica*, 10, (1994), 379-403.
- [16] Junger M. and Mutzel P. 2-Layer Straightline Crossing Minimizations: Perf. of Exact and Heuristic Alg. (1997).
- [17] Kapur N. *Cell Placement and Minimization of Crossing Numbers*. M.S. Thesis, North Carolina State University, Raleigh, NC, 1998.
- [18] Gergel N., Craft S., and Lach J. Modeling QCA for Area Minimization in Logic Synthesis. *Great Lakes Symposium on VLSI*, (2003), 60-63.
- [19] Gary Bernstein, "Quantum-dot Cellular Automata: Computing by Polarized Systems", Proc. Design Automation Conference, 2003.