Global Placement for Quantum-dot Cellular Automata Based Circuits

Jean Nguyen[†], Ramprasad Ravichandran[‡], Sung Kyu Lim[†], and Mike Niemier[‡]
[†] School of Electrical and Computer Engineering
[‡] College of Computing
Georgia Institute of Technology
{jnguyen@ece, raam@cc, limsk@ece, mniemier@cc}.gatech.edu

Abstract

Quantum Cellular Automata (QCA) has been proposed as an alternative architecture to CMOS and in principle should permit the implementation of ultra lower-power, nano-scale logic circuitry working at teraflop frequency. QCA is based on a new paradigm for encoding binary logic into electronic circuitry, where binary 1s and 0s are mapped to spatial configurations of electrons rather than magnitudes of electronic currents. The layout rules for QCA based circuits are radically different from those of CMOS based circuits, and design automation tools for QCA circuit layout are hard to find. This paper discusses the first automatic global placement algorithm for QCA-based circuits. We divide the QCA global placement process into zone partitioning and zone placement, and identify the constraints and objectives that are unique to QCA-based circuits as opposed to the conventional CMOS VLSI.

1. Introduction

For almost the past seven decades, the design of digital electronic computers has been dominated by two enduring (and most successful) ideas - using binary numbers to represent information mathematically, and physically representing those binary numbers as the "on" and "off" state of a current switch. First proposed by Konrad Zuse, the current switch has provided virtually every structure and function needed for a modern computer that uses semi-conductor transistors. Still, despite the technological and commercial success of CMOS, the Semiconductor Research Association roadmap still predicts that to ensure future generations of chips and fabrication processes, problems must be overcome for which the roadmap states, "there are no known solutions." [1] Nevertheless, despite dire predictions, the potential to technologically solve or alleviate many of the above problems facing CMOS is not unrealistic. However, while researchers may be successful in besting the laws of nature, the laws of economics may prove to be insurmountable. With a fabrication plant projected to cost approximately 200 billion dollars in the year 2015 [2], CMOS scaling may just be too expensive to continue. Thus, one need not be committed to a particular forecast to see the growing importance of developing alternative approaches that would permit the scaling of computational electronics down to the ultimate limits of molecular dimensions.

One approach to such scaling is the nano-scale quantum-dot cellular automata (QCA) concept that uses only one of the two ideas that make up Zuse's paradigm – specifically using a binary representation of information, but replacing the current switch with a cell having a bi-stable charge configuration. A QCA device usually consists of 2 or 4 quantum dots and either 1 or 2 excess electrons respectively. One configuration of charge represents a binary '1', the other a binary '0', but no current flows into or out of the cell. In the transistor paradigm, the current from one device charges a gate on the next device, the interconnect between them, and thus turns the device on or off. In the QCA paradigm, the field from the charge configuration of one device alters the charge configuration of the next device. Remarkably, this basic device-device interaction is sufficient to allow the computation of any Boolean function, and also forms the interconnection mechanisms. If a clocking potential is added which modulates the energy barrier between charge configurations, general purpose computing becomes possible with very low power dissipation.

At present, there is much ongoing research concerning nano-scale *devices* geared for computation; but most work is exactly that – *device* work. By in large, only the simplest circuits and systems comprised of such devices (which are the desired end result), have been proposed, let alone simulated and built. Furthermore, virtually all existing circuits and systems for any emergent device have been proposed by the device engineers themselves whose background lies largely in the physical sciences, not engineering. In order to obtain a more realistic outlook for systems in QCA, for the past six years QCA has included a systems-level research component. It began when QCA's device physics was sufficiently advanced and specifically focused on ideal architectures for the technology, computationally interesting, yet implementable circuits and systems, and Mead-Conway-esq design rules to help abstract away lower-level device physics to help system designers become more involved with this work. Overall goals included comparing

projections for implementable QCA to projections for end of the line CMOS in the context of the same system-level tasks, providing an infrastructure for more complex designs as technology matures, and using architecture, systems, and circuits work to help drive device development to get to working nano-systems sooner.

In the context of the above goals, this paper will describe a set of tools that will help to generate computationally interesting, yet implementable circuits in QCA, and significantly expand QCA's existing systems-level infrastructure. Specifically, it will discuss the first physical layout automation algorithm that generates global placement for QCA-based circuits. Our QCA global placement is divided into *zone partitioning* and *zone placement*. The purpose of zone partitioning is to partition the input circuit so that a single potential modulates the inner-dot barriers in all of the QCA cells in each zone. The zone placement step takes as input a set of zones with each zone a clocking label obtained from zone partitioning. The output of zone placement is the best possible layout for arranging the zones on a 2 dimensional chip area.

The remainder of this paper is organized as follows: Section 2 present background information on QCA. Section 3 presents the problem formulation. Section 4 and 5 respectively presents our zone partitioning and zone placement algorithms. Section 6 presents our experimental results, and we conclude our paper in Section 7.

2. Preliminaries

2.1. QCA Devices

A high-level diagram of a "candidate" four-dot metal QCA cell appears in Figure 1. It depicts four quantum dots that are positioned to form a square. Exactly two mobile electrons are loaded into this cell and can move to different quantum dots by means of electron tunneling. Coulombic repulsion will cause "classical" models of the electrons to occupy only the corners of the QCA cell, resulting in two specific *polarizations*. These polarizations are configurations where electrons are as far apart from one another as possible, in an energetically minimal position, without escaping the confines of the cell. Here, electron tunneling is assumed to be completely controllable by potential barriers that can be raised and lowered between adjacent QCA cells by means of capacitive plates parallel to the plane of the dots [3]. It is also worth noting that in addition to these two "polarized" states, there also exists a decidedly non-classical *unpolarized* state. Briefly, in an unpolarized state, inter-dot potential barriers are lowered to a point that removes the confinement of the electrons on the individual quantum dots, and the cells exhibit little or no polarization as the wave functions of two electrons smear themselves across the cell [4].

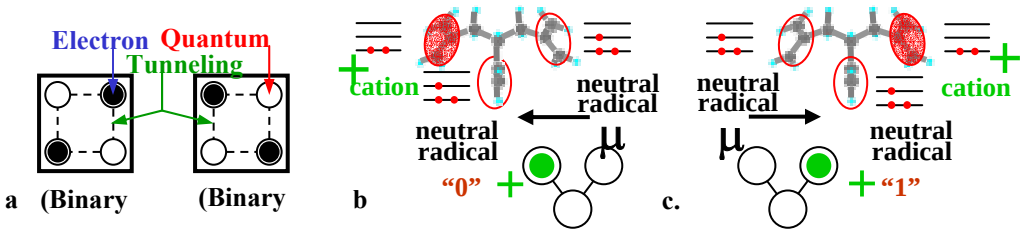

Figure 1. Schematic representation of metal-dot (a) and molecular (b-c) QCA cells.

It is also possible to construct QCA cells from individual chemical molecules [5]. In contrast to metal-dot cells, the small size of molecules (on the order of 1-5 *nm*) means that Coulomb energies are much larger, so room temperature operation is possible. At the molecular scale, the coupling and electrostatic interaction between molecular devices is on the electron Volt scale. The thermal energy present at room temperature is on the order of 0.025 electron Volts, indicating that errors caused by thermal energies of the environment in which a molecular QCA cell is operating will not cause the cell to propagate the wrong binary information [6]. In addition, the power requirements and heat dissipation of QCA are low enough that high-density molecular logic circuits and memory are feasible. In contrast to lithographic device fabrication techniques, which always introduce variations in device characteristics, each molecular cell can be made exactly identical using chemical synthesis. Information about specific molecular QCA implementations is readily available in literature [6,7,8], with 2-"dot", 3-"dot", and 4-"dot" implementations all under investigation. A schematic device is shown in Figure 1. Finally, while molecules are seen as a more natural implementation for QCA, experiments continue in both veins of research. Metal-dot QCA experiments have been used to prototype molecular QCA devices, and ideas transfer between implementations. Given this, QCA's logic functionality will be explained in terms of "generic" 4-dot cells.

2.2. QCA Logical Elements

Majority gate: 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 Figure 2(a). 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 the three input cells. Here, the electrostatic repulsion between the electrons in the three input cells, and the electrons in the device cell will be at a minimum.

Wires: One way of moving data from point *A* to point *B* in a QCA circuit is with a 90-degree wire. (The wire is called "90-degrees"as the cells from which it is made up are oriented at a right angle). The wire is a horizontal row of QCA cells and 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. Here, as a binary signal propagates down the length of the wire, it alternates between a binary 1 and a binary 0 polarization. By placing a 90-degree cell between and adjacent to two 45-degree cells, both the original signal value and its complement can be obtained (in the latter case without the use of an explicit inverter circuit). As the majority voting function can be reduced to an AND or OR function (by setting an input to a 0 or a 1), QCA's logic set is functionally complete. Finally, QCA wires possess the unique property that they are able to cross in the plane without the destruction of the value being transmitted on either wire as shown in Figure 2(d). This property holds only if the QCA wires are of different orientations (i.e. a 45-degree wire crossing a 90-degree wire). However, it is most important as at present, all layout is assumed to be two-dimensional.

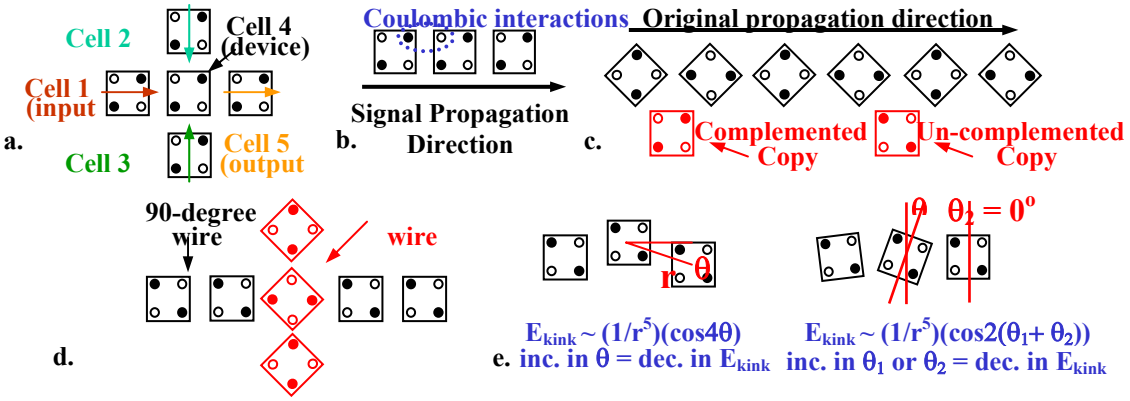

Figure 2. A QCA majority gate (a), 90-degree wire (b), 45-degree wire and ripper (c), wire cross (d), and error relationships (e).

Error: QCA cells do not have to be perfectly aligned to transmit binary signals correctly. "Wires" have some tolerance for fabrication errors caused by misalignment, improper cell rotation, improper cell-spacings, etc. Although imperfect, a wire might still transmit a binary value successfully. External energy (defined as E_{kink}) can cause a cell in a wire or a system to switch into a mistake state. More specifically, the kink energy is the amount of energy that will excite a cell into a mistake state (or create a "kink" in the wire). Referring to Figure 2(e), the kink energy for off-center cells is proportional to $(1/r^5)\cos(4\theta)$. Thus, as the distance between cells increases, the kink energy will decrease indicating that a smaller amount of external energy could excite a cell into a mistake state which is undesirable. If the angle of off-centeredness between cells increases, E_{kink} will also decrease. Disorder can also arise because of cells with improper rotation. In this situation, kink energy is proportional to $(1/r^5)\cos(2(\theta_1+\theta_2)$ where θ_1 and θ_2 are the angles at which two cells are rotated (see Figure 2(e) for more detail).

2.3. The QCA clock

When compared to CMOS, another striking difference in circuits and systems of QCA cells is the mechanism used to clock them. In standard CMOS, the clock is generally considered to be a signal that precisely controls the time at which data bits are transferred to or from memory elements. It typically has two phases: high and low. In QCA, the clock is not a separate wire or port that would be fed into a circuit like any other signal. Rather, it is typically viewed to be an electric field that controls barriers within a QCA cell, which in turn controls whether or not excess charge in a QCA cell can represent a binary 1 or 0.

QCA's clock was first characterized by Lent, et. al. as having 4 phases (see Figure 3). During the first clock phase (*switch*), QCA cells begin un-polarized with inter-dot potential barriers low. During this phase barriers are raised, and the QCA cells become polarized according to the state of their drivers (i.e. their input cells). It is in this clock phase, that actual switching (or computation) occurs. By the end of this clock phase, barriers are high enough to suppress any electron tunneling and cell states are fixed. During the second clock phase (*hold*), barriers are held high so the outputs of the subarray that has just switched can be used as inputs to the next stage. In the third clock phase, (*release*), barriers are lowered and cells are allowed to relax to an unpolarized state. Finally, during the fourth clock phase (*relax*), cell barriers remain lowered and cells remain in an unpolarized state [4].

Individual QCA cells need not be clocked or timed separately. However, a physical array of QCA cells can be divided into *zones* that offer the advantage of mutli-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. When a circuit is divided into different zones, each zone may be clocked differently from others. In particular, this difference is important when discussing neighboring, or physically adjacent zones. Such a clocking scheme allows one zone of QCA cells to perform a certain calculation, have its state frozen by the raising of inter-dot barriers, and then have the output of that zone act as the input to a successor zone. It is this mechanism that provides the inherent self-latching associated with QCA.

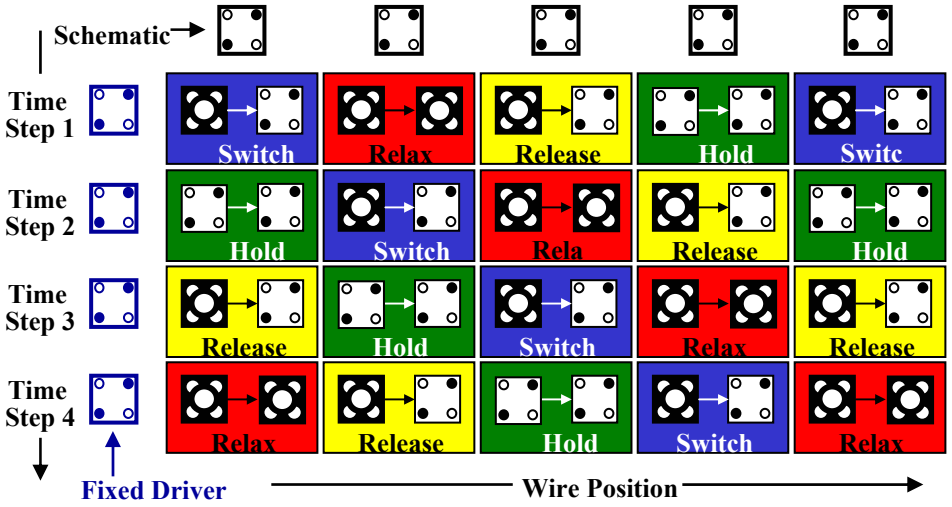

Figure 3: An example of a pipelined QCA wire. Each cell is clocked individually.

In a molecular implementation of QCA, the four phases of a clock signal would most likely take the form of time-varying but repetitious voltages applied to silicon wires embedded underneath some substrate to which QCA cells were attached. Every fourth wire would receive the same voltage at the same time [9]. Neighboring wires see delayed forms of the same signal. The charge and discharge of the embedded silicon wires will move the area of activity (i.e. computation or data movement) across the molecular layer of QCA cells with computation occurring at the "leading edge" of the applied electric field. Computation moves across the circuit in a continuous "wave" [7,9]. Still, it is important to stress that whether or not a four-phase model or a computational wave model of QCA's clock is used, the work presented here will be applicable to both. The end goal in either clocking model is to ensure that QCA data signals arrive at the proper place (i.e. a majority gate) at the proper time. Assuming a four-phase clock, this is accomplished by balancing clocking zones. Assuming a computational wave, we must balance silicon wires.

3. Problem Formulation

Our QCA global placement is divided into *zone partitioning* and *zone placement*. The purpose of zone partitioning is to partition the input circuit so that a single potential modulates the inner-dot barriers in all of the QCA cells in each zone. Unless we group QCA cells into zones and provide zone-level clock signals, each individual QCA cells needs to be clocked. In addition, the latency of the pipe-lined QCA system will be unacceptable. Therefore, zone partitioning simplifies the clock signal distribution dramatically and improves the latency. However, since the delay of the biggest partition determines the overall clock period, the size of the partition needs to be determined carefully.

In addition, the 4-phase clocking imposes a strict constraint on how to perform partitioning. When a clocking zone is in switch phase, all of its immediate predecessors need to be in hold phase while all of it immediate successors should be in relax phase. In other words, every directed path in the partitioned network should follow "switch-relax-release-hold" sequence. The zone placement step takes as input a set of zones with each zone a clocking label obtained from zone partitioning. The output of zone placement is the best possible layout for arranging the zones on a 2 dimensional chip area.

3.1. Zone Partitioning

The QCA-based circuit is represented with a directed acyclic graph (DAG) $G(V,E)$. Let P denote the partitioning of V into K non-overlapping and non-empty blocks. Let $G'(V',E')$ be a graph derived from P, where V' is a set of *logic blocks* and E' is a set of cut edges based on P. A directed edge $e(x,y)$ is called *cut* if x and y belong to different blocks in P. Two paths p and q in G' are *reconvergent* if they diverge from and reconverge to the same blocks. Let $l(p)$ denote the *length* of a reconvergent path p in G'. Then $l(p)$ is defined to be the number of cut edges along p. The following set of constraints exists in QCA zone partitioning problem:

1. clocking constraint: all reconvergent paths starting at the same block should have the same length.
2. acyclicity constraint: there should be no directed cycle among the blocks in G'.
3. logic capacity constraint: the area of each logic block should be within the user specified range.
4. wire capacity constraint: the area of each wire block should be within the user specified range.

If P violates the clocking constraint, we correct this problem by inserting *wire blocks* to G'. Each wire block does not contain any logic QCA cells but wiring QCA cells only (Section 4.1 discusses in more detail how wire block corrects the clocking problem). The QCA Zone Partitioning Problem seeks a legal partitioning solution that minimizes the amount of inter-zone wires, wire blocks required, and latency. Latency is measured by the total number of inter-zone edges along the longest path in G'. Area capacity constraint requires that the number of QCA cells per clock zone should be kept under a threshold in order to make sure that cells reach the ground state instead of remaining at the excited state. Wire capacity constraint requires that the amount of wires included in each wire block should be kept minimal in order to increase the probability of successful QCA switching and minimize the clock period. Clocking and acyclicity constraints are related to implementing correctly 4-stage pipelined QCA circuits, where every directed path in the partitioned network follows the "switch-relax-release-hold" sequence.

3.2. Zone Placement

Assume all blocks in P have the same area. In such a case, the placement of P becomes a geometric embedding of G' onto $m \times n$ grid, where each logic/wire block is assigned to a unique location in the grid. For a given edge $e(B_1,B_2)$ in E', *edge distance* $d(e)$ denotes the Manhattan distance between B_1 and B_2. The goal of QCA Zone Placement Problem is to minimize the total number of wire crossing, sum of all edge distances (= wirelength), and final placement area.

Even though theoretical physics tells us that QCA wires with different cell orientations can cross in the plane with no disruption on either value being transmitted on either wire, such a configuration is not seen as realizable in near-to-midterm QCA experiments. This problem will be explained in the context of molecular QCA cells that are viewed to be the most natural and promising QCA implementation mechanism. One process envisioned for creating systems of QCA cells is as follows: first, a molecular QCA cell would be engineered that will pack and assemble properly on a self-assembled monolayer (SAM) on top of a silicon surface. Second, I/0 structures would be constructed lithographically. Third, tracks would be etched into the self-assembled monolayer (SAM) on top of a silicon surface with EBL. Finally, the resulting "chip" would then be dipped into a bath of QCA cells for self-assembly with devices binding to the etched tracks. Currently, the simple tasks of making QCA cells attach to some substrate, in some deterministic pattern, with the same cell rotations is non-trivial. Allowing for selective rotation would only complicate this process even more. Consequently, systems with few (or no) wire crossings are viewed as ideal.

Considering wirelength in general, while it is to some extent a function of implementation technology, wire length is also largely a function of kink energy. As an example, consider a linear array of N cells that form a wire that we want to transmit a logical 1. The ground state for this configuration would be all of the cells switching to the same polarization as that of the driving cell—namely a line of cells in the logical '1' polarization. The first excited (mistake) state of this array will consist of the first m cells polarized in a representative binary 1 state and N-m cells

in the binary 0 state. The excitation energy of this state E_k is the energy required to introduce a "kink" into the polarization of the wire. This energy is independent of where the kink occurs (i.e. the exact value of *m*). As the array *N* becomes larger, the kink energy E_k remains the same. However, the entropy of this excited state increases as there are more ways to make a "mistake" in a larger array. When the array size reaches a certain size, the free energy of the mistake state becomes lower than the free energy of the correct state meaning that a value will not propagate. A complete analysis reveals that the maximum number of cells in a single array is given by $\exp(E_k/k_BT)$ [4]. Thus, given an E_k of 300 meV (reasonable as will be seen the discussion of molecular experiments), k_b (1.38×10^{-23} J/K), close to room temperature operation (300K), and that $1 \text{ J} = 1.6 \times 10^{-19}$ eV, arrays of cells on the order of 10^5 are not unreasonable.

4. Zone Partitioning

4.1. Overview of the Approach

A significant issue in QCA circuit is the proper synchronization of the asynchronous clocks applied to each partition to ensure proper signal propagation through the circuit. To guarantee this, the partition level network should be acyclic and all reconvergent paths should be of equal length. An example of a set of reconvergent paths is shown in Figure 4(a). In Figure 4(a), the paths *S-A-B-T*, *S-C-T* and *S-D-E-F-T* are three reconvergent paths with unbalanced lengths. This poses a bigger problem in QCA circuits since all the partitions are asynchronously clocked. Let us consider the path *S-A-B-T* at a time instant *t*. If partition *S* is in the *switch* phase of the clock cycle, *A* will be in the *relax* phase of the cycle, while *B* and *T* will be in the *release* and *hold* stages respectively. Now taking into consideration the path *S-C-T* at the same instant *t*, partitions *C* and *T* will have to be in the *relax* and *release* stages respectively. Similarly considering the *S-D-E-F-T* path, *T* will be in *switch* stage if partition *A* is in *switch* stage too. Hence it is impossible for all signals to properly propagate into *T* given this layout. In order to solve this problem, *wire blocks* are inserted in all paths shorter than the longest reconvergent path. Wire blocks are feedthrough partitions that posses their own clocking phase. Figure 4(b) shows the same circuit as in Figure 4(a) but with the wire blocks inserted to solve the clocking inconsistency problem in reconvergent paths.

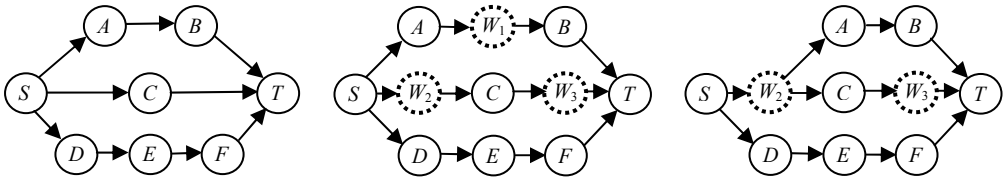

Figure 4. Illustration of unbalanced reconvergent path problem and wire block insertion

In Figure 4(b), nodes W_1, W_2 and W_3 are the wire blocks. Now if *S* is in *switch* stage, partitions *A,*W_2 and *D* are in the *relax* stage, while nodes W_1,*C* and *E* are in *release* stage and nodes *B*, W_3 and *F* are in the *hold* stage. Hence, *T* can remain in the *switch stage* and now all signals from *A* are propagated into *T* correctly. Figure 4(c) shows the same reconvergent path problem solved with fewer wire blocks, where wiring blocks share the resources together.

The purpose of zone partitioning is to partition the input circuit so that a single potential modulates the inner-dot barriers in all of the QCA cells in each zone. The goal is to minimize the amount of inter-zone wires, wire blocks required, and latency while satisfying the clocking, acyclicity, and logic/wire capacity constraints as mentioned in Section 3.1. The two-way partitioning problem with cutsize minimization is NP-hard already, and adding the minimization of wire blocks required and latency under the clocking, acyclicity, and wire capacity constraints increases the complexity of the problem even higher. Thus, our iterative improvement based approach is to start from a legal solution that satisfies the acyclicity and logic capacity constraint and attempts to minimize the wire blocks required.

Let $lev(p)$ denote the longest path length from the input partitions (= partitions with no incoming edges) to partition *p*, where the path length is the number of partitions along the path. Then $wire(e)$ denotes the total number of wire blocks to be inserted on an inter-partition edge *e* to resolve the unbalanced reconvergent path problem (= clocking constraint of the QCA zone partitioning problem). Simply $wire(e) = lev(y) - lev(x) - 1$ for $e=(x,y)$, and the total number of wiring blocks required without resource sharing is $\sum wire(e)$. It is important to note that the direct

minimization of $\sum wire(e)$ is time-consuming since the computation and update of $lev(p)$ requires $O(n)$ path analysis (Section 4.3 discusses this analysis in more detail). In addition, it is also difficult to predict the number of wire blocks required when resource sharing is allowed. We note that balancing the reconvergent path lengths has positive impact on minimizing the number of wire blocks required to fix any clocking violation. Thus, our heuristic approach is to minimize the *variance* of *wire*(e) among all inter-zone edges so that the reconvergent path lengths are balanced. Then during our post-process, we fix any remaining clocking problem by inserting and sharing wire blocks while satisfying the wire capacity constraints.[1]

4.2. Cell Gain Computation

First, the cells are topologically sorted and evenly divided into a number of partitions (p_1, p_2, ... p_k). The partitions are then level numbered using a breadth-first search. Then the acyclic FM partitioning algorithm [12] is performed on adjacent partitions p_i and p_{i+1}. Following the acyclic FM algorithm, cells are sorted into bucket based on their gain. A maximum pointer *maxptr* is maintained for each bucket which points to the cell with the highest gain. One at a time, a cell from the bucket with the highest *maxptr* is removed from the bucket and moved to the other partition provided all constraints are met. Constraints that must be met include logic capacity and acyclicity. The logic capacity criterion is that each partition must have an area A such that $(1-\varepsilon)r \le A \le (1+\varepsilon)r$, where ε represent the area skew. The acyclicity criterion does not allow edges to be directed from partition p_{i+1} to p_i. To enforce this criterion, the type of move is first checked. If the cell is moving to a partition with a higher level number p_{i+1} and all of the cell's fanout is located in a partition p_r, where $r > i+1$, then the move is allowed. Also, if the cell is moving to a partition with lower level number p_{i-1} and all of the cell's fanin is located in p_r where $r < i-1$, then the move is allowed. If the cell is found to violate the acyclicity constraint, the cell remains in its original partition and a move is not allowed. After a cell has been moved or forced to remain in its original partition, the cell is locked and removed from the bucket. Moves are made until there remain more cells that are unlocked. After all moves have been made, the best partition from that pass is taken as the output of the pass. The best partition is the one who has the lowest cost $\alpha \times cutsize + \beta \times variance$. Multiple passes are performed on two partitions p_i and p_{i+1} until there is no more improvement on the cost. Then, this acyclic bipartitioning is performed on partitions p_{i+1} and p_{i+2}, then p_{i+2} and p_{i+3}, and so on.

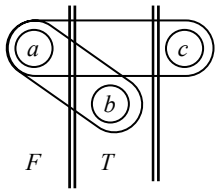

Figure 5. Illustration of cutsize gain under terminal propagation.

Each cell gain is given by $\alpha \times g_c + \beta \times g_v$ where α and β are user specified integers that will bias the different gains. For $\alpha>0$ and $\beta=0$, zone partitioning algorithm is performing acyclic FM algorithm with moves based solely on reducing cutsize. A net is *cut* if it spans more than one partition, and *cutsize* counts the total number of cut nets. Cutsize gain g_c is the amount by which the cell changes the current cutsize if the cell were to be moved to the other partition. The $O(n)$ algorithm for initializing and updating cutsize gain is followed from the acyclic FM algorithm. The zone partitioning algorithm begins with an initial partitioning solution of k partitions such that terminal propagation is considered. In case we partition p_i and p_{i+1}, a net is *external* if it contains cells that are not in p_i nor p_{i+1}. If we perform partitioning between F and T as in Figure 5, net *a*-*c* is external and *a*-*b* is not. To account for terminal propagation, only the initialization part of the cutsize gains from acyclic FM algorithm needs to be modified, and the updating of cutsize gains remains the same. Figure 5 shows an example where FM algorithm would compute the gain of cell *a* to be $g_c = +1$, but our initialization algorithm computes the cell gain to be $g_c = +2$. Figure 6 shows the pseudo-code for computing the initial cutsize gain under terminal propagation.

[1] The latency minimization resembles the performance-driven partitioning problem where the delay along the longest path is minimized. In addition, cyclic dependency among QCA zones is hard to handle. Our ongoing works try to address these problems.

```
------------------------------------------------------------------------
Algorithm INITIALIZE_Gc
FOR each free cell i DO
     FOR each net n on cell i DO
          IF F(n) = 1 AND T(n) = 0
               IF (i is in lowest partition n spans & T<F) THEN gc(i)++
               IF (i is in lowest partition n spans & T>F) THEN gc(i)--
               IF (i is in highest partition n spans & T>F) THEN gc(i)--
               IF (i is in highest partition n spans & T<F) THEN gc(i)++
           ELSE
                 IF F(n) = 1 THEN gc(i)++
                 IF T(n) = 0 THEN gc(i)--

Algorithm INITIALIZE_Gv
FOR each free cell i DO
     FOR each net n on cell i DO(i)
          IF (i is a minimum anchor & T>F) THEN gv(i)++
          IF (i is a minimum anchor & T<F) THEN gv(i)--
          IF (i is a maximum anchor & T>F) THEN gv(i)--
          IF (i is a maximum anchor & T<F) THEN gv(i)++

Algorithm UPDATE_Gv
FOR each net n on the base cell which was an anchor point DO
     IF (base cell is a maximum anchor & F>T)
          THEN gv(i)-- for the minimum anchor cell of net n
     IF (base cell is a minimum anchor & F>T)
          THEN gv(i)-- for the maximum anchor cell of net n
     IF (base cell is minimum anchor & the new external net length = 0)
          THEN gv(i)-- for maximum anchor cell of net n
     IF (base cell is maximum anchor & the new external net length = 0)
          THEN gv(i)-- for minimum anchor cell of net n
------------------------------------------------------------------------
```

Figure 6. Zone partitioning algorithm.

Variance gain g_v is related to the amount by which the cell changes the variance among *wire*(*e*) for all inter-zone edges. As discussed earlier, *wire*(*e*) denotes the total number of wire blocks to be inserted on an inter-partition edge *e* to resolve the unbalanced reconvergent path problem. After the partitions are level-numbered, *anchor* cells for each net is identified. A cell is defined to be an anchoring cell if it is located in the partition with the smallest or largest level number for an external net and it is the only cell located in that partition. Only moves of anchoring cells will cause a change in the length of that external net. The change in variance depends on whether a cell move will be increase or decrease the length of the external net. For a cell move that will be increasing the length, change in variance is $\Delta\sigma_i = -(2d \times ln + d^2)/j + (2d \times \sum ln + d^2)/j^2$, where d represents the absolute value of the change in length and j accounts for the number of external nets in the circuit. For a cell move that will be decreasing the length, change in variance is $\Delta\sigma_d = (2d \times ln - d^2)/j - (2d \times \sum ln - d^2)/j^2$. Due to the use of buckets for cell gain, we represent the gain by the number of nets a cell acts as an anchoring point for. Cells that are not anchoring points for any nets have $g_v = 0$. Initialization algorithm for g_v is shown in Figure 6.

Figure 7 shows cell *c*, *b*, *e* to be anchoring cells, while cell *a* and *d* are not. Furthermore, when partitioning between partition p_1 and partition p_2, $g_v(e) = +1$ and $g_v(b) = +1$. When partitioning between partition p_2 and partition p_3, $g_v(b) = -1$ and $g_v(c) = +2$. Similar to updating cutsize gain, the variance gain of only neighbor cells needs to be updated. The updating is done only when the moved cell was an anchor point and only to neighboring anchor cells connected by the net which this moved cell served as anchor point. For cell moves which were not anchoring points, the updating of anchor for nets may need to be performed. Furthermore, gain is updated if the base cell is an anchor and it makes a move which makes the length of the external net zero. The algorithm for updating $g_v(i)$ is shown in Figure 6.

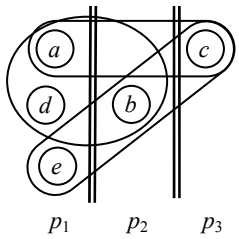

Figure 7. Illustration of anchor cells and variance gain computation

4.3. Updating Level Numbers and Anchors

Movement of a single cell can possibly change *lev*(*p*), the level number of a partition *p*. Therefore every time a cell move is made, we check to see if this cell move affects the level number. There are two ways levels can change: an inter-zone edge is newly introduced or completely removed. In Figure 8(b), cell *a* in Figure 8(a) is moved from partition *A* to *B*, thereby creating a new inter-partition edge. This in turn changes the level of all downstream partitions. In Figure 8(c), cell *a* in Figure 8(a) is moved from partition *A* to *C*, thereby removing the inter-partition edge between *A* and *C*. This again changes the level of all downstream partitions. For updating the level, we maintain a *maxparent* for each *p* so that the level number of *p*'s parent is *lev*(*p*)–1. *lev*(*F*) is defined as the level number of the "from block" of a cell *c* and *lev*(*T*) is defined as the level number of the "to block" of *c*. In the first case where a new inter-partition edge is created, *lev*(*T*) is updated if *lev*(*F*) ≥ *lev*(*T*) after the cell move. In this case, the new *lev*(*T*)=*lev*(*F*)+1. Then, we recursively update the *maxparent* and levels of all downstream partitions. The *maxparent* for partition *C* was changed from *A* to *B* in Figure 8(b), and *lev*(*C*) now becomes *lev*(*B*)+1 = 2. This in turn requires the level number of all downstream nodes to change. In the second case where an existing inter-partition edge is removed, the *maxparent* again needs to be update. The *maxparent* for partition *C* was changed from *A* to none in Figure 8(c), and *lev*(*C*) now becomes *lev*(*C*)=0.

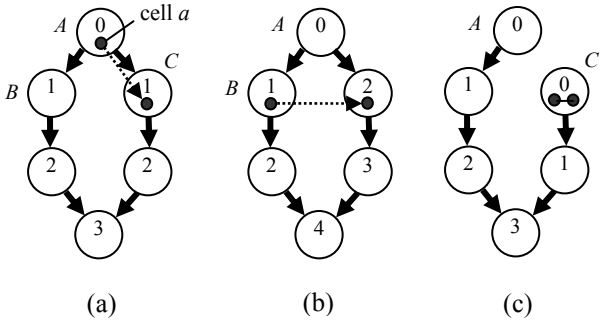

Figure 8. Illustration of partition level update.

In addition to a cell move affecting the level number, the cell move can also affect the anchor for the net(s) to which it is attached. With a move for an anchoring cell for net *n*, if the cell moves to a partition whose neighboring cell from net *n* is located in, then the anchoring cell no longer becomes an anchor for that corresponding net. If a non-anchoring cell *a* moves to a partition with level number greater than or less than the level number of the anchoring cell *b*, then *a* becomes the new anchor and *b* is no longer the anchor. If *a* moves to a partition with level number equal to *b*, then neither *a* nor *b* acts as an anchor for *n*.

4.4. Wire Block Insertion

Our zone partitioning heuristic minimizes the *variance* of *wire*(*e*) among all inter-zone edges so that the reconvergent path lengths are balanced. Then during our post-process, we fix any remaining clocking problem by inserting and sharing wire blocks while satisfying the wire capacity constraints. The pseudo-code of the algorithm is shown in Figure 9.

```
---------------------------------
Algorithm CLOCK-PARTITIONS(G,V,E)
CLOCK(SUPERNODE) = -1
Q.ENQUE(SUPERNODE)
BFS-MARK(G,SUPERNODE)
BIN = POPULATE-BIN(G,E)

Algorithm BFS-MARK(G,Q)
N = Q.DEQUE()
S = Set of fanout neighbors of N
While (S not empty)
     A = S.POP()
     If (LAST-PARENT(A) == N)
          CLOCK(A) = CLOCK(N)+1
          Q.ENQUE(A)
BFS-MARK(G,Q)

Algorithm POPULATE-BIN(G,E)
While (E not empty)
     N = E.pop()
     S = CLOCK(N.SRCNODE)
     T = CLOCK(N.SNKNODE)
     While (S + 1 < T)
          S = S+1
          BIN[S] = {BIN[S],E}
Return BIN
---------------------------------
```

Figure 9. Pseudo-code for wire block insertion.

The input to this algorithm is the set of partitions and inter-partition edges. First, a super-source node is inserted in the graph whose fan-out neighbors are the original sources in the graph. This is done to ensure that all sources are in the same clocking zone. Then the single–source longest path is computed for the graph with the super-source node as the source, and every partition is assigned a clocking level based on its position in the longest path from the source. For a graph with E inter-partition edges, this algorithm runs in exactly E iterations. In the next stage of the algorithm, any edge connecting partitions separated by more than one clock phase is marked and the edge is added to an array of bins at every index where a clocking level is missing in the edge. For instance, if an edge is found between two partitions say A and B, where A having clocking level 3 and B with clocking level 7, the edge $\{A,B\}$ is added to bin numbers 4,5 and 6. The pseudocode of this phase is given in algorithm POPULATE-BIN. This is done so that wire block at same clocking levels can be merged to give a solution similar to the solution in Figure 4(c).

Next, the number of wire block in each bin is calculated based on a predetermined capacity for the wire blocks. This capacity is calculated based on the width of each cell in the grid. Then the inter-partition edges are distributed amongst the wire block filling one wire block to full capacity before filling the next. Though, it might seem that a better solution would be to evenly distribute the edges to all the wire blocks in the current level (to minimize power), this is not so. This is because the wire blocks with the most number of feedthroughs are placed closer to the logical blocks in the next stage. This minimizes wirelength, and hence the number of wire crossings. It could also potentially reduce the critical delay in the circuit. A pictorial representation of the clocking and layout of the circuit is given in Figure 10. All blocks, wire and logical, at the same height are in a single clocking zone.

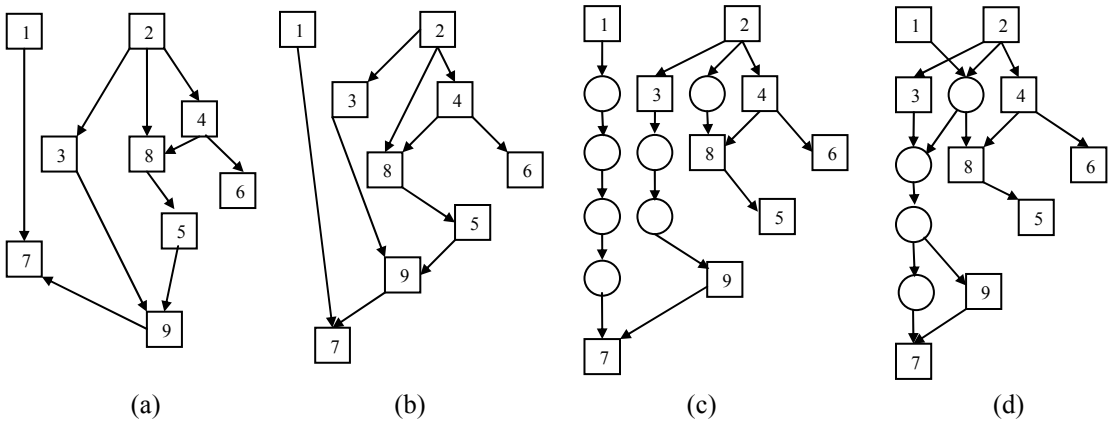

Figure 10. Pictorial overview of the first two stages. The square blocks denote logical partitions, and the circles are wire blocks. All blocks, wire and logical, at the same height are in a single clocking zone.

5. Zone Placement

Similar to CMOS circuits, a good placement is a key step in solving the automated layout problem in QCA circuits. In this phase, the zone partitioning result obtained from the previous stage is placed in a grid based layout, each grid cell occupied by either a logic or wire block. The global placement phase has to minimize wire length and wire crossing under clocking constraints. First, we assign blocks in the partitioned network onto grid cells while considering the given aspect ratio constraint. Then the partitions are rearranged within their clocking zones to optimize on the various objectives listed above. An analytical solution and a simulated annealing based solution for this stage are implemented and compared.

5.1. Grid Placement

As evident from Figure 10(d), the resulting partitioned network after the wire block insertion satisfies $lev(x)=lev(y)+1$ for each inter-partition edge $e=(x,y)$. In this case, a bipartite graph exists for every pair of neighboring clocking levels. Therefore, our grid placement problem is to embed this graph onto an $m \times n$ grid with a given aspect ratio. The logical blocks (obtained from the partitioning stage) and the wire blocks (obtained from our post-process) are placed on an $m \times n$ grid with a given aspect ratio α and skew δ. The individual cell dimensions and the column widths are kept constant to ensure scalability and manufacturability of this design since clocking lines have to be laid beneath the QCA circuit with great precision and proper timing. The partitions are laid out on the grid, with the cells belonging to the first clocking zone occupying the leftmost cells of the first row of the grid, and the next level occupying the leftmost cell of the next row and so on till the m^{th} row. The next level of cells is placed again on the m^{th} row just right to the rightmost cell amongst the m placed rows. The next level of cells are placed in the $m-1^{th}$ row and rest of the cells are placed in a similar fashion till the first row. This process is repeated until all the cells are placed. Figure 11 demonstrates an example. The *dummy nodes* are white space that is introduced because of variations in the number of wire and logic blocks among the various clocking levels.

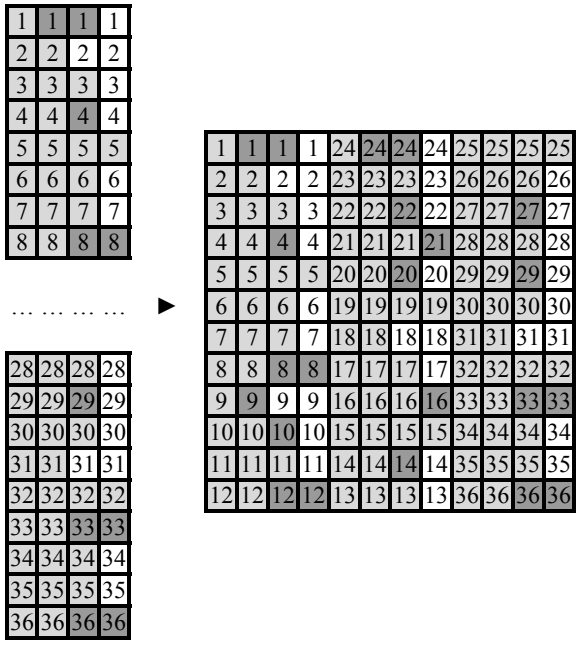

Figure 11. Grid layout of the zone partitions. Light gray, dark gray, and white grid cell respectively denotes the logic, wire, and dummy blocks. Numbers in the cells denote the clocking level the cell belongs to.

5.2. Wire Crossing and Length Minimization

At the end of grid placement, we have a 2D array of cells arranged by clocking level. During our next phase, the blocks are reordered within each clocking level to minimize the inter-partition wire lengths and wire crossings. Two classes of solutions were applied to minimize the above objectives, (i) an analytical solution that uses a *weighted barycenter method,* and (ii) *Simulated Annealing*. Additionally, simulated annealing was applied with the analytical solution as the initial solution.

The maximum wire length between any two partitions in the grid determines the clock frequency for the entire grid since all partitions are clocked separately. For example, if the wire length between all nodes has a maximum of 30 nm, but for one particular inter-partition edge where the wire length is 60 nm, that could reduce the clock speed for the entire circuit by a factor of 2, since the clock period must be doubled so that this 60 nm wire can be clocked properly. For the first and last rows (where the inter-partition edges are between partitions in two different columns), maximum wirelength was given more priority since maximum wirelength at these end zones can be twice as bad as the maximum wirelength between partitions on the same column. This is illustrated in Figure 12. The edge {*A*,4} in Figure 12 (a) has a Manhattan distance of 4 while the same edge in Figure 12 (b) has a Manhattan distance of 8.

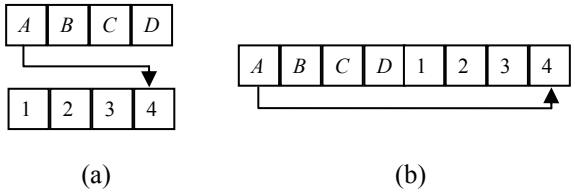

Figure 12. Illustration of the end zone effects of placing two clocking zones on the same row but subsequent columns

5.2.1. Analytical Solution

A widely used method for minimizing wire crossing (introduced by Sugiyama et al. [12] and Carpano [13]) is to break the graph into *k* layers and then the vertices within a layer are permuted to minimize wire crossings. This method perfectly renders itself in this problem since we need to only consider the latter part of the problem (since the clocking constraint yields us the *k* layers). But, even in a 2-layer graph, minimizing wire-crossings is NP-hard

[14]. Some of the common heuristics used to solve the one-sided crossing minimization are the barycenter heuristic [12], the split heuristic [15], the greedy-switch heuristic [17], median heuristic [16], stochastic heuristic [17], and the assign heuristic [18]. Amongst these heuristics, the barycenter heuristic has been found to be the best heuristic in the general case for this class of problems [19].

The *barycenter method* proposed by Sugiyama et al. [12] involves sorting the nodes in each level of a bipartite graph based on a number called the *barycenter* which is a measure of where the connections to the next level of the graph are most concentrated. Every node in the variable layer gets a relative position based on its barycenter number. The modified version of the heuristic was used to accommodate for *edge weights*. The edge weights represent the number of multiple inter-partition edges connecting the same pair of partitions. The heuristic can be summarized as below.

$$barycenter(v) = \frac{\sum_{n \in N} edgeweight(n) * position(n)}{\sum_{n \in N} edgeweight(n)}$$

where v is the vertex in the variable layer, n is the neighbor in the fixed layer, N is the set of all neighbors in the fixed layer.

5.2.2. Simulated Annealing

Simulated Annealing is a generalization of a Monte Carlo method which was originally introduced by Metropolis et al. [20]. This concept is based on the way metals when melted and slowly cooled, recrystalize to reach a lowest energy state. Applying Simulated Annealing to solve combinatorial problems was introduced by Kirkpatrick et al. [21]. Simulated annealing has been used earlier in solving automated placement for CMOS circuits [23,14,15,16]. A *move* or *perturbation* in our algorithm is constituted by randomly choosing a level in the graph, and then swapping two randomly chosen partitions in that level in order to minimize the total wirelength and wire crossing. If the new cost function is better than the old, then the move is conditionally accepted. However, if the new move is worse, then the solution is accepted with a probability based on the Boltzmann distribution [22]. Updating the wirelength and wire crossing takes $O(n)$ if not done carefully. IN our approach, we initially compute the wirelength and wire crossing and incrementally update these values after each move so that the update can be done in $O(1)$ time. This speedup allows us to explore more number of candidate solutions and obtain better quality solutions.

The initial temperature was set so that half of the *moves* were accepted. This was done by making 1000 random swaps in the initial solution, and storing the change (Δ_i) in the *cost function* in every successive swap. The swaps that lead to a positive change in the *cost function* were averaged (δ) and the initial temperature, T_i was set according to the following equations: $C = \gamma \times \text{tot\_crossing} + \delta \times \text{tot\_wirelength}$, $\Delta_i = C_i - C_{i-1}$, where γ & δ are empirically chosen parameters such that the magnitude of tot_crossing is similar to that of tot_wirelength. Then, $\varepsilon = (\sum \Delta_i)/N$, and $T_i = \varepsilon/\ln(2)$. At every temperature, a fixed number of moves was performed and the temperature was reduced by a factor r=0.87. This number was empirically chosen as well. The final temperature was set to $T_i \times r^{200}$.

6. Experimental Results

Our algorithms are implemented in C++/STL, compiled with gcc v2.96 with –O3, and run on Pentium III 746 MHz machine. The benchmark set consists of seven circuits from ISCAS89 [28] and five circuits from ITC99 [27] suites. Our goal is to use perform global placement for these circuits based on QCA structure. The statistical information of benchmark circuits is shown in Table 1. We provide the number of gates, PIs, POs, FFs, nets, and partitions for each circuit. The number of partitions is determined in such a way that each partition contains 100±10 cells (=logic capacity). Wire blocks are allowed to contain 200 incoming inter-zone connection (= wire capacity).

Name	# Gates	# PI	# PO	# FF	# Nets	# Part
b14_opt	5401	32	299	245	5678	59
b15_opt	7092	37	519	449	7577	80
b17_opt	22854	37	1511	1414	24305	258
b20_opt	11979	32	512	490	12501	130
b21_opt	12156	32	512	490	12678	131
b22_opt	17351	32	725	703	18086	188
s13207	8027	31	790	669	8727	95
s15850	9786	14	684	597	10397	110
s35932	16353	35	2048	1728	18116	201
s38417	22397	28	1742	1636	24061	258
S38584	19407	12	1730	1452	20871	226
S5378	2828	36	212	163	3026	32
S9234	5597	36	250	211	5844	60

Table 1. Benchmark circuit characteristics.

6.1. Zone Partitioning Results

Table 2 shows our zone partitioning results, where we report the cutsize, variance, number of dummy nodes, and wire blocks for each circuit. We compare the result of acyclic FM and QCA zone partitioning. QCA partitioning differs in acyclic FM partitioning in that cell moves are based on variance gains (α=0 and β=1), whereas acyclic FM partitioning makes cell moves based solely on cutsize gains (α=1 and β=0). With QCA zone partitioning algorithm, there is a 20% improvement in cutsize at the cost of 6% increase in runtime. We compared our cutsize results to state-of-the-art CMOS VLSI multiway partitioning algorithm [12] and verified that our results are comparable. Furthermore, a slight but consistent improvement in variance translates to again slight but consistent improvement in number of dummy nodes wire blocks. Our main objective in minimizing the variance was to reduce the number of wiring nodes. But, we observe that there is no guarantee that reduction in variance will always decrease the number of wire blocks as evident from s5378 result.

Our goal during zone partitioning was to balance the reconvergent path lengths through variance minimization among *wire*(e) so that the number of wire blocks required to fix any clocking violation is minimized. As discussed earlier in Section 4.3, however, direct minimization of wire blocks is time-consuming since the computation and update of partitioning level requires $O(n)$ path analysis upon each move. During our post-process (= wire block insertion), we fix any remaining reconvergent path problem by inserting and sharing wire blocks while satisfying the wire capacity constraints. Again, it is hard to predict and optimize the number of wire blocks added during this step.

The number of dummy nodes—the white spaces resulting from grid placement—can be reduced by taking into account the balance between the number of blocks per clock level during partitioning. Similar to the minimization of the number of wire blocks, the balance can be improved by performing a post processing phase, where cell moves are made from a block in a congested clock level to its neighboring block in a less congested clock level. Furthermore, the balance can also be improved by creating more partitions in a congested but narrow clock level.

6.2. Zone Placement Results

Table 3 shows our zone placement results, where we report the placement area, wirelength, and wire crossing for each circuit. We compare the analytical solution to simulated annealing with random start and analytical start. Looking at total wirelength, we see a slight decrease in simulated annealing with analytical start as compared to the analytical solution by itself, and an 87 % decrease in simulated annealing with random start as compared to the analytical solution. Considering the total number of wire crossings, we see a slight increase in simulated annealing with analytical start as compared to the analytical solution by itself, and slight increase in simulated annealing with random start as compared to the analytical solution. We observe a positive correlation between total wire length and number of inter partition wire crossings. Also, simulated annealing with an analytical start does not seem to be able to jump out of the neighborhood whose valley is given by the analytical solution.

	Acyclic FM Partitioning				QCA Zone Partitioning			
Circuits	Cut	Var	Dummy nodes	Wire blocks	Cut	Var	Dummy nodes	Wire blocks
b14_opt	2948	177	151	138	2566	175	168	127
b15_opt	4839	462	220	260	4119	468	144	256
b17_opt	16092	2603	1565	1789	13869	2669	1616	1710
b20_opt	6590	796	641	519	6033	819	642	518
b21_opt	6672	877	599	560	6141	869	622	557
b22_opt	9473	1952	1146	1097	8518	1949	1158	1098
s13207	2708	445	143	138	1541	442	144	137
s15850	3023	523	257	183	2029	522	254	181
S35932	7371	1919	875	1014	5361	1734	734	1035
S38417	9375	3275	757	784	5868	3285	775	773
S38584	9940	4109	1319	1155	7139	3881	1307	1095
s5378	1206	55	34	30	866	52	34	30
s9234	1903	165	99	81	1419	154	104	76
Ave	6318	1335	600	596	5036	1309	592	584
Ratio	1.00	1.00	1.00	1.00	0.80	0.98	0.99	0.98
Time	14646				14509			

Table 2. Zone partitioning results.

		Analytical		Simulated Annealing Random start		Simulated Annealing Analytical start	
ckts	area	wire length	wire xing	wire length	wire xing	wire length	wire xing
b14_opt	20x17	81395	67378	23055	67378	81395	67378
b15_opt	20x24	59283	90125	34190	90125	59283	90125
b17_opt	69x52	3014967	346894	305843	345267	3014967	346894
b20_opt	36x36	414367	165218	99221	166324	414367	165218
b21_opt	36x37	140282	172197	100359	172197	140282	172197
b22_opt	48x50	1091091	230812	188256	230812	1091091	230812
s13207	18x21	28381	9413	28381	9413	5682	8274
s15850	24x23	81678	16925	11543	14088	82586	15183
S35932	45x44	1313068	64933	78434	68603	1313068	64933
S38417	42x43	493973	54864	48372	54864	493973	54864
S38584	55x48	1500043	102297	110211	80326	1500043	102297
s5378	10x10	3746	10098	2590	9688	3746	10098
s9234	15x16	15212	11462	5661	11462	15212	11462
Ave		633653	103278	79701	101581	631977	103057
Ratio		1.00	1.00	0.13	0.98	1.00	1.00
Time		23		661		609	

Table 3. Zone placement results.

7. Conclusions and Ongoing Works

In this paper we formulated the QCA global placement problem and presented the first algorithm. We are currently working on cell placement stage, where each individual QCA cell is placed while honoring our global placement results. Our ongoing work for partitioning includes further reducing the number of wiring nodes and number of dummy nodes inserted. We are also trying to construct zone partitioning solution in such a way that a separate post-process to insert wiring block is not necessary. We are working on an improved version of the analytical solution for global placement, which takes into account wire lengths when assigning edges to wire blocks. The motivation is that longer wires should use the wire blocks further away from the logical blocks to minimize crossing. In the simulated annealing solution we are looking at swapping edges between wiring blocks as a legal move instead of swapping blocks alone. Further, while placing blocks onto rows in the grid, we are investigating a better way of compacting the logical and wire blocks to minimize white space (= dummy nodes).

Reference

[1] Semiconductor Industry Association, "The National Technology Roadmap for Semiconductors", 2000.

[2] P. Rutten and M. Tauman and H. Bar-Lev and A. Sonnino, "Is Moore's Law Infinite? The Economics of Moore's Law", *Kellogg TechVenture Anthology*, pp 1-28, 2001.
[3] P.D. Tougaw and C.S. Lent, "Logical Devices Implemented Using Quantum Cellular Automata", *Journal of Applied Physics*, Vol 75, pp 1818, 1994.
[4] Craig S. Lent and P. Douglas Tougaw, "A Device Architecture for Computing with Quantum Dots", *Proceedings of the IEEE*, vol 85, pp541, 1997.
[5] C.S. Lent, "Molecular Electronics: Bypassing the Transistor Paradigm", *Science*, Vol 288, pp 1597-1599, 2000.
[6] Marya Lieberman, Sudha Chellamma, Bindhu Varughese, Yuliang Wang, Craig Lent, Gary H. Bernstein, Gregory Snider and Frank C. Peiris, "Quantum-dot Cellular Automata at a Molecular Scale", *Annals of the New York Academy of Science*, vol 960, pp 225-239, 2002.
[7] Craig S. Lent, Gregory L. Snider, Gary Bernstein, Wolfgang Porod, Alexei Orlov, Marya Lieberman, Thomas Fehlner, Michael Niemier and Peter Kogge, *Quantum-Dot Cellular Automata*, 2003.
[8] A. Aviram, "Molecules for Memory, Logic, and Amplification", *Journal of the American Chemical Society*, vol 110, pp 5687-5692, 1988.
[9] K. Hennessy and C.S. Lent, "Clocking of molecular quantum-dot cellular automata", *Journal of Vacuum Science and Technology*, vol 19, pp 1752-1755, 2001.
[10] C. M. Fiduccia and R. M. Mattheyses. "A Linear-Time Heuristic for Improving Network Partitions". *Proceedings of the Design Automation Conference*, pp 174-181, 1982.
[11] removed for blind review
[12] K. Sugiyama, S. Tagawa and M. Toda, "Methods for Visual Undestanding of Hierarchical System Structures", *IEEE Trans. Syst. Man, Cybern*., SMC-11 (1981) 109 – 125
[13] M. J. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Transactions on Systems*, Man, and Cybernetics, 10(11):705-715, 1980.
[14] P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 10:379--403, 1994.
[15] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatoria*, 21-A:89--98, 1986.
[16] P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 10:379--403, 1994.
[17] S. Dresbach. A new heuristic layout algorithm for DAGs. In U. Derigs and A. B. . A. Drexl, editors, *Operations Research Proceedings* 1994.
[18] C. Catarci. The assignment heuristic for crossing reduction. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(3), 1995
[19] M. Jünger and P. Mutzel. 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms, 1997
[20] Metropolis,N., A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, "Equation of State Calculations by Fast Computing Machines", *J. Chem. Phys*.,21, 6, 1087-1092, 1953..
[21] Kirkpatrick, S., C. D. Gelatt Jr., M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, 220, 4598, 671-680, 1983
[22] P.J.M. van Laarhoven, E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, Kluwer Publ., Dordrecht, 1987
[23] C. Sechen., *VLSI placement and global routing using simulated annealing*. Boston: Kluwer Academic, 1988.
[24] C. Sechen and K. Lee. An Improved Simulated Annealing Algorithm for Row-Based Placement. *IEEE Intl. Conf. on CAD (ICCAD)*, pages 478-481, November 1987.
[25] W. Swartz and C. Sechen. New algorithms for the placement and routing of macro cells. *In Proc. 27th Design Automation Conference*, pages 336-339, 1988.
[26] W. Sun and C. Sechen. Efficient and Effective Placement for Very Large Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3):349-359, March 1995.
[27] http://www.cad.polito.it/tools/itc99.html
[28] http://www.cbl.ncsu.edu