Node duplication and routing algorithms for quantum-dot cellular automata circuits

W.J. Chung, B. Smith and S.K. Lim

Abstract: Quantum-dot cellular automata (QCA) is a novel computing mechanism that can represent binary information based on the spatial distribution of electron charge configuration in chemical molecules. QCA circuit layout is currently restricted to a single layer with very limited number of wire crossings permitted. Thus, wire crossing minimisation is crucial in improving the manufacturability of QCA circuits. We present the first QCA node duplication and routing algorithms for wire crossing minimisation. Our duplication algorithm named fan-out tolerance duplication (FTD) explores node duplication in conjunction with node placement using K-layered bipartite graphs (KLBG). FTD successfully removes additional crossings at the cost of increased area and allows flexible tradeoff between area and wire crossing. Our routing algorithm, namely cycle breaker (CB), constructs a modified vertical constraint graph (VCG) to enforce additional vertical relation for wire crossing reduction. We formulate and provide a heuristic solution for the weighted minimum feedback edge set problem to effectively remove cycles from the VCG. As a result, FTD and CB achieve wire crossing results that are very close to theoretical lower bound and outperform the conventional algorithms significantly.

1 Introduction

Nanotechnology and devices will have revolutionary impact on the computer-aided design (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 realised in metal [1], or with chemical molecules [2]. A wealth of experiments have been conducted with metal-dot QCA, with individual devices [1], logic gates [3, 4], latches [5], and clocked devices [5], all having been realised. This advancement is followed by various recent efforts in developing CAD tools for QCA based circuits and systems [6–8].

In this paper, we present the first QCA node duplication and routing algorithms for wire crossing minimisation. The goal is to focus on the following undesirable design schematic characteristics associated with a near-to-midterm buildability point of QCA circuits: large amounts of deterministic device placement, long wires, clock skew, and wire crossings. Our duplication algorithm attempts to: (1) rearrange logic gates or nodes to reduce wire crossing, (2) duplicate minimal number of nodes to further reduce wire crossing, (3) provide a mechanism for flexible wire crossing against area tradeoff, (4) further minimise area by identifying and removing redundant nodes and (5) create shorter routing paths to logical gates to reduce signal skew. Our channel routing algorithm constructs a modified vertical constraint graph (VCG) to enforce additional vertical relation for wire crossing reduction. We formulate and provide a heuristic solution for the weighted minimum feedback edge set problem to effectively remove cycles from the VCG. As a result, we achieve wire crossing results that are very close to theoretical lower bound and outperform the conventional algorithms significantly.

The remainder of the paper is organised as follows: Section 2 presents problem formulation. Section 3 presents our QCA duplication and post-process algorithms. Section 4 presents our QCA channel routing algorithm. We provide experimental results in Section 5 and conclude in Section 6.

2 Preliminaries

2.1 Issues on wire crossing in QCA circuits

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 realisable in near-to-midterm QCA experiments. This problem can 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

IEE Proceedings online no. 20050278

doi:10.1049/ip-cds:20050278

Paper first received 5th October 2005 and in revised form 23rd February 2006

W.J. Chung is with the Department of Electrical Engineering, Stanford University, 161 Packard Building, Standford, CA 94305, USA

B. Smith and S.K. Lim are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, 777 Atlantic Drive NW, Atlanta, GA 30332, USA

E-mail: limsk@cce.gatech.edu

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. Consequently, systems with few (or no) wire crossings are viewed as ideal.

2.2 Problem formulation

QCA circuits are built from the following components: QCA device [1], QCA logic gates [4], QCA wires [3], and QCA clocks [9] [Note 1]. The following set of constraints exists during QCA circuit layout: (1) the terminal constraint: the I/O terminals are located on the top and bottom boundaries of each logic block, (2) the signal direction constraint: the signal flow among the logic QCA cells needs to be unidirectional-from the input to the output boundary for each time zone [5]. The signal direction is caused by QCA's clocking scheme, where an electric field E created by underlying CMOS wire is propagating uni-directionally within each block. Thus, placement and routing need to be done in such a way to propagate the logic outputs in the same direction as E. In order to balance the length of wires, we construct a k-layered bipartite braph as follows:

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

Signal skew will determine how fast the circuit can be clocked. Owing to the unidirectional (from top to bottom) clocking scheme, we attempt to minimise skew at individual layers. Thus we introduce a new term, layer signal skew:

Definition 2: Layer signal skew: the difference between the maximum and minimum latency of all signals going through layer m, where latency of a particular signal through m is defined as the time from when the signal leaves m to when it reaches layer $m+1$.

The formal definition of QCA placement with duplication and QCA channel routing are as follows:

Definition 3: QCA placement and duplication: we seek a KLBG placement and duplication of individual logic gates so that area, wire crossing, and wirelength are minimised. A duplication is valid if (i) the duplicated node exists in the same layer of the KLBG as the original node, and (ii) all input signals of the original node are also present at the duplicated node.

Definition 4: QCA channel routing: the goal is to finish connection among the terminals in every two adjacent layers of the KLBG so that the total wire crossing and channel width are minimised.

2.3 Previous work

The crossing number problem [10] is a problem of determining, for a given integer K, whether a graph G can be embedded in a 2-D plane with K or fewer pairwise crossings of the edges (not including the intersections of the edges at their common endpoints). The crossing minimisation problem is that of embedding a graph in the plane with the minimum number of edge crossings. This problem arises in the graph drawing application and CMOS VLSI layout. The crossing number problem was proven to be NP-complete [10], and several heuristics have been proposed [11]. The bipartite crossing minimisation problem is a special case of the crossing minimisation problem, where the two sets of nodes in a bipartite graph are placed in two parallel lines. This problem has been proved NP-complete [12], and several heuristics exist [13].

In CMOS VLSI layout, wire crossing distribution and minimisation problem has been solved for global routing [14]. Cell duplication has been especially popular in circuit partitioning for area and performance optimisation [15]. Some recent effort focuses on performing duplication for placement enhancement [16] and FPGA timing optimisation [17]. However, cell duplication has never been applied to minimise wire crossing. Many channel routing algorithms exist that attempt to minimise the channel width [18] or crosstalk [19] recently. However, wire crossing has not been the focus for channel routing due to the availability of multiple routing layers.

Our duplication algorithm utilises the existing Barycenter algorithm [12] as a post-process. But the Barycenter algorithm itself cannot handle duplication. Our routing algorithm extends the existing YK algorithm [18] to optimise the new wire crossing objective in addition to the traditional track width objective.

3 Wire-crossing minimisation

3.1 Overview

The only method for reducing wire crossing on a single plane, without increase in area, is rearranging the nodes. The problem of determining the position of each node for the absolute minimum wire crossings is NP-complete; the same problem for a single layer was already shown to be NP-complete [12]. Thus we employ Barycenter [12], an effective heuristic algorithm with simulated annealing for this purpose. To further reduce wire crossing, we then apply our fan-out tolerance duplication (FTD) algorithm. To minimise the increase in area, critical nodes and regions are prioritised for duplication.

3.2 Barycenter algorithm

In a KLBG, Barycenter traverses all the layers from output to input. For each node in the current working layer, a 'weight' is assigned based on the average column of its fan-outs. The nodes are then rearranged according to their weights, from the lowest (left-most column) to the highest (right-most column). By doing so, the algorithm attempts to place nodes directly above their fan-out nodes, reducing the average distance to its outputs and thus eliminating unnecessary wire crossings.

It is in our interest to reduce as many wire crossings with Barycenter for there is no cost in area. Thus we attempt to improve on the heuristics of the algorithm by employing simulated annealing. Since the weight of a node is determined by its fan-outs, there are no weights assigned for the nodes in the output layer as they do not have outgoing edges. Therefore, Barycenter assumes that the nodes in the output layer are already in their optimal positions. Experiments confirmed if these nodes were rearranged, Barycenter produced a different number of wire crossings. Simulated annealing was performed on the arrangement of the output nodes, where the cost function was the number of wire crossings after applying Barycenter.

The option of reverse-Barycenter has also been considered, where we traversed the graph from the input to the output layer and used average fan-ins for the weight. In this

Note 1: Detailed discussions of these building blocks are omitted in this paper owing to the page limit. Interested readers are referred to the respective reference

case, the input layer would be assumed to have been placed optimally. Our experiments indicated that reverse-Barycenter performed much worse than Barycenter, in terms of wire crossings.

3.3 Fan-out tolerance duplication algorithm

The motivation of the fan-out tolerance duplication (FTD) algorithm is from the general observation that the longer a wire that connects two nodes, the higher chance it has of causing wire crossings. Thus FTD focuses on identifying long wires and reducing their lengths. The bipartite nature of a KLBG allows us to estimate wire length with the number of columns an edge spans over. If a particular node had only one output to the adjacent layer, the wire crossings that it may cause can be eliminated by placing the node in the same column as its destination node. This is demonstrated in Fig. 1*a*, where the wire length has been reduced to zero as it does not reach over to another column. However in (*b*), if a node had multiple fan-outs, and the destination nodes were apart, no matter where this node is placed the total length of its fan-outs and the wire crossings they cause will remain constant.

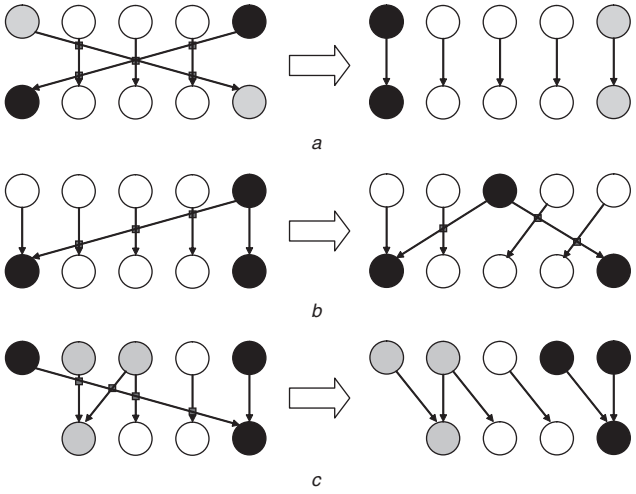

Fig. 1 *Three different scenarios of the Barycenter algorithm*
a, *c* The algorithm successfully reduces wire crossings
b When it fails

We present the FTD algorithm in what follows. The KLBG is traversed from output to input. Nodes that have split fan-outs, where destination nodes are apart farther than the threshold, are duplicated. This parameter is calculated with the tolerance specified as a percentage of the total number of columns. After all duplications have been made at a particular layer, it must then be put through Barycenter again as the weights of the nodes have changed. An example of FTD with threshold of 2 is shown in Fig. 2.

Fan-out tolerance duplication algorithm

1: $threshold = tot{:}column * tolerance$
2: for ($i = tot.layer - 1$ downto 0)
3: for (each node j in layer i)
4: for (each fan-out k of j)
5: for (each fan-out l of j except k)
6: if ($|col(k) - col(l)| > threshold$)
7: duplicate_node(j, i, l);
8: barycenter(i);

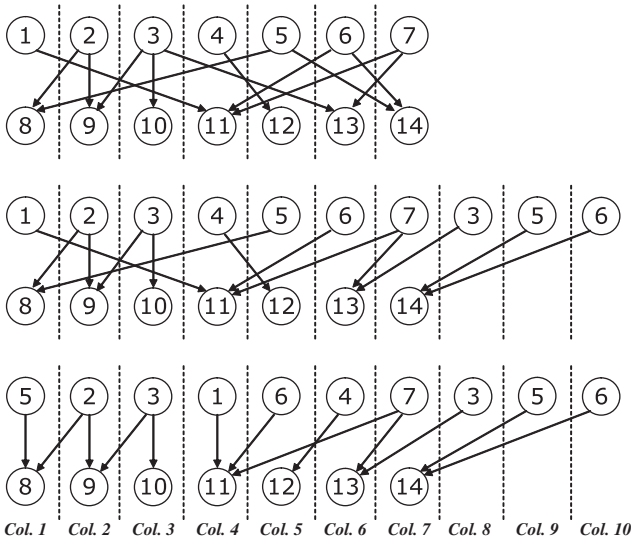

Fig. 2 *Example of the FTD algorithm with the threshold value of 2*
All nodes whose fan-outs spread out more than the threshold are duplicated. After duplication, the layer is rearranged with Barycenter

duplicate_node(j, i, l)
1: $j' =$ copy of node j in layer i excluding fan-outs;
2: place j' in layer i;
3: remove fan-out l of node j;
4: add fan-out l to node j';

By controlling the tolerance of the FTD algorithm, it is possible to generate an entire range of wire crossing against area tradeoff points. If the tolerance is 0%, the resulting circuit will be wire crossing free as all fan-outs in all nodes would have been removed via node duplication. Alternatively, for a tolerance of 100%, no duplication will take place and we are left with the same post-Barycenter circuit. Any intermediate tolerance will produce a corresponding area and wire crossing between the two extremes.

The primary weakness of FTD is that as a node is duplicated, all of its inputs must be duplicated as well. Thus, if the node had a lot of fan-ins, the duplicated fan-ins may create more wire crossings than what was removed. The algorithm would handle these new crossings when it processes the next layer, but the duplication rate would increase sharply. Hence, for circuits that have a high fan-in count, wire crossing reduction can only be expected after a significant number of duplications or increase in area.

3.4 Priority fan-out tolerance duplication

In the process of trading off wire crossing reduction for area or additional QCA cells, we would like to prioritise duplication for nodes that cause the highest number of crossings. However, to generate a list of the most problematic nodes is troublesome as the removal of a single node will affect the wire crossings caused by other edges. But at the layer level, the problem scales down to identifying layers that contain most wire crossings. Thus, we can prioritise layers instead of individual nodes for our duplication strategy.

Since FTD is applied to a post-Barycenter circuit, understanding the characteristics of Barycenter can help us determine the most effective layers to prioritise FTD. We note that Barycenter uses the average fan-out column as its weight; hence it handles fan-ins well but fails on fan-outs.

This is depicted in Fig. 1*c* and *b*. Observing the fan-in and fan-out behaviour of many benchmark circuits, it can be said that most of the circuits generally take upon a 'diamond' shape shown in Fig. 3. Region *a* is the fan-out region where input signals spread out to the entire width of the logic block. Layers in this region generally have more fan-outs than fan-ins. In region *b*, processed signals converge to produce the output signals, resulting in fewer fan-outs. The height ratio between *a* and *b* varied among circuits but in most cases, *a* was significantly shorter than *b*, which then explained why reverse-Barycenter performed poorly. Reverse-Barycenter has the opposite characteristics of Barycenter and its weakness was being applied to the larger region of the circuit.

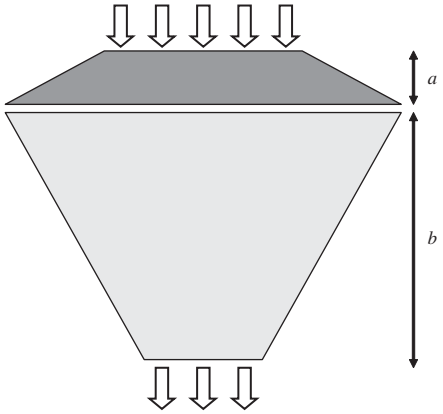

Fig. 3 *General fan-in, fan-out behaviour of circuits*
Region *a* is the fan-out region where input signals get spread throughout the circuit. Region *b* is the fan-in region where processed signals converge towards the output

3.5 Post optimisation of area and signal skew

The first post-process after node duplication is to reduce the number of nodes or area as much as possible. Owing to the characteristic of the FTD algorithm where all fan-outs of a single node are identified and possibly duplicated back to back, there are frequent cases of identical nodes existing adjacent to each other. In such cases, consecutive identical nodes may be merged into a single node to reduce area. An example of neighbour-merging is shown in Fig. 4. Neighbour-merging does not generate additional wire crossings and may even reduce them in certain cases. Thus the process is always performed as it is always beneficial.

Our second post-process is layer folding. Figure 5 illustrates the effect of our layer-folding. For each and every logic gate that has been folded onto a new layer, the previous layer must provide bypass wires. Naturally, there must be bypass wires in the new layer for all logic gates remaining in the original layer. Despite the insertion of

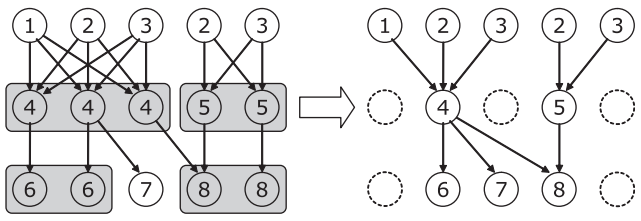

Fig. 4 *Example of neighbour-merging, which removes redundant nodes*
Area and wire crossing may be reduced from this process

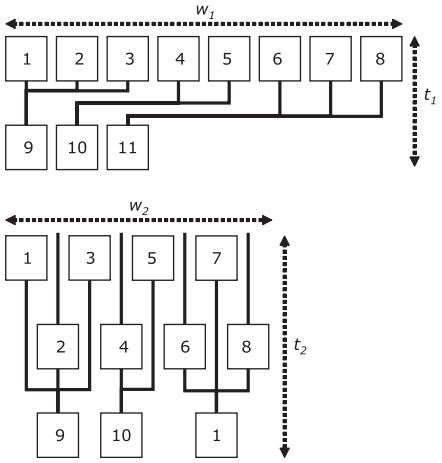

Fig. 5 *Physical implementation of layer-folding*
Logic gate blocks are larger than bypass wires, thus the width of a single layer can be reduced by folding blocks onto a new layer. The width and layer signal skew has been reduced at the cost of more wires and increased area

bypass wires, owing to the fact that every component remains within the same column (but in different layers), layer-folding guarantees no additional wire-crossing. Also note that every other node should be folded down to minimise the length of the bypass wires.

The width of a QCA wire is significantly less than the width of a logic gate. Thus it is possible to decrease the layer width as a logic gate is exchanged for a bypass wire. With a thinner circuit, the maximum latency would be shortened as the possible horizontal shift range is reduced. Therefore, the difference between the maximum and minimum latency will decrease. Alternatively, layer-folding increases the number of layers, lengthening the time required for the circuit to produce the output. Insertion of bypass wires also increases the overall area of the circuit.

The number of logic gates to move down to a new layer is determined by the target width. The width after layer-folding for a layer with n logic gates can be calculated as $W = L(n - s) + s$, where L represents the average width ratio between a QCA logic gate to a wire and s is the number of gates folded down. There also exists an absolute minimum width achievable, where the folded layer would only contain a single logic gate (and bypass wires). Hence the target width must be a value between the current width and the absolute minimum. Owing to the fact that layer-folding is costly, as an entirely new layer is inserted to the KLBG per folding, the post-process should only be applied to layers that do not meet the layer signal skew constraint.

4 QCA channel routing

4.1 Overview

After the QCA gates have been placed into a KLBG, each channel in the KLBG must be routed. First, the channel terminals are traversed from left to right, and each contact edge is added to the VCG (vertical constraint graph) [18] (line 2). Since the VCG must be acyclic, cycles in the contact edges are removed by empty column insertion and vertical doglegs (line 3). Once the VCG is acyclic, crossing edges are added to reduce wire crossing (line 4). If cycles exist in the VCG, contact edges are intelligently removed until the VCG is once again acyclic (line 5). Lastly, we use left-edge first (LEF) algorithm [20] to assign tracks to the nets and finish routing the channel (line 6). This entire process is

applied to each channel of the KLBG from top to bottom to route the entire circuit (line 1).

4.2 Crossing edge insertion

In general, the vertical constraint graph (VCG) [18] is used in graph-based channel routers to capture the relations among the terminals. In case there exist cycles in VCG, we insert doglegs to break them [21] so that the given netlist is routable. In this paper, we introduce two kinds of edges to be used in our VCG: (i) contact edge: a directed edge in the VCG resulting from the contact (or terminal) positions in the channel. For each pair of top and bottom contacts in the same column of the channel, an edge is created to order the top contact's net above the bottom contact's net to prevent their vertical tracks from overlapping. (ii) crossing edge: a directed edge in the VCG added to orient the nets for optimum wire crossing minimisation. For a given pair of nets (n_1, n_2), a crossing edge $e = (n_1, n_2)$ is added if routing n_1 above n_2 results in a more wire crossing reduction than routing n_1 below n_2. Each crossing edge is assigned a weight equal to the number of wire crossings it reduces.

We note that wire crossing can only occur between nets which overlap horizontally, i.e. if an edge exists between them in a horizontal constraint graph (HCG) [18]. Also, the crossing count between any arbitrary pair of horizontally overlapping nets is determined by their vertical ordering. For example, consider Net 1 and Net 2, which overlap horizontally in Fig. 6. If Net 1 is placed below Net 2, as in Fig. 6*a*, then they will cross twice: a crossing for each top terminal of Net 1 and for each bottom terminal of Net 2 located in this region of overlap. Likewise, if Net 2 is placed below Net 1, as in Fig. 6*b*, they will cross once: a crossing for each bottom terminal of Net 1 and for each top terminal of Net 2 located in this region of overlap. A similar relationship exists between Nets 1 and 3, but for the opposite order. For any pair of nets, one vertical orientation will result in as many or fewer crossings as the other vertical orientation. The orientation with the least crossings is the optimum vertical orientation of these nets to reduce crossing. If the orientation of every net with every other net is enforced in this manner, then the resulting crossing count will be reduced as much as possible.

To reduce crossing, we use VCG to orient the nets with vertical relationships that reduce the crossing counts between pairs of nets. We add previously defined crossing edges between nets to enforce the vertical relationship that results in the minimum number of crossing between them. For each pair of Net 1 and Net 2 that overlaps horizontally, we compare the amount of wire crossing reduced by routing Net 1 above Net 2, and vice versa. We then add a crossing edge to enforce the vertical net orientation that results in more wire crossing reduction. Each crossing edge is assigned a weight equal to the number of wire crossings saved by orienting the nets according to that vertical orientation. An original VCG is shown in Fig. 6*c*, and a modified version is shown in Fig. 6*d*. The resulting channel, shown in Fig. 6*e*, results in the fewest wire crossings. The theoretical lower bound of wire crossing for each circuit is equal to the sum of the minimum values of wire crossing for each pair of nets. In the same way, the upper bound of crossing is equal to the sum of the maximum values for each pair. These lower and upper bounds were calculated and appear in Table 1.

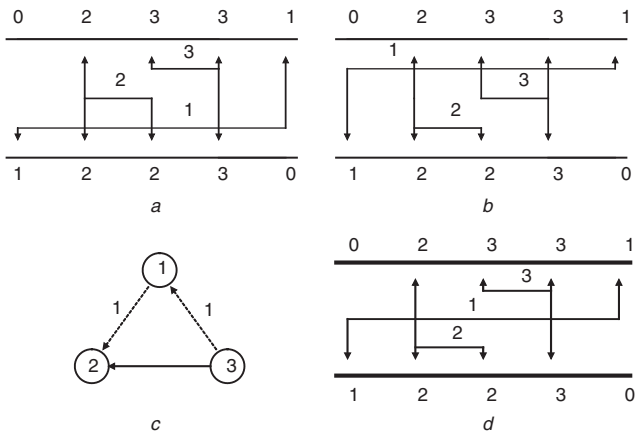

Fig. 6 *Overlap of Net 1 and Net 2*
a Wire crossing between n_1 and n_2 is 2
b Wire crossing between n_1 and n_2 is 1
c Two crossing edges (dashed lines) are inserted to the VCG, where the edge weights denote the wire crossing reduction
d Minimum wire crossing solution based on *c*

Table 1: KLBG construction

	Di-graph		KLBG		
ckt	♯node	♯edge	♯layer	♯col	♯bypass
adder	99	130	19	62	600
k2	317	2893	4	215	35
c1908	938	1523	42	221	4946
pair	1140	2157	20	666	5720
i8	1397	4707	10	1712	6585
c3540	1741	2961	49	512	4753
t481	2089	6824	12	2834	5080
i10	2978	5600	56	1152	24 955
s5378	3239	4427	35	623	8520
c7552	3827	6252	45	806	11 216
s9234	6094	8221	60	1089	24 288
b15_ opt	8097	16 375	47	2711	81 897
s13207	9517	12 031	61	1824	48 971
s15850	11 081	1434	84	1631	75 483
b21_opt	13 190	27 198	76	3931	113 763
b17_opt	25 816	53 133	46	9102	256 803
big3	37 381	63 535	288	12 497	235 3419
big1	40 564	60 172	41	12 989	343 306
big4	44 649	66 205	273	14 752	214 4751
big2	62 225	109 161	470	16 663	594 8331

4.3 Optimal cycle breaking

If the VCG is not acyclic after the addition of crossing edges, then the cycles must be removed before track assignment can take place. An illustration is shown in Fig. 7*a*. Since the contact edges cannot be removed without invalidating the solution, we remove crossing edges until the VCG is once again acyclic. The weight of crossing edges represents wire crossing reduction, so our goal is to minimise the total weight of the crossing edges removed to make VCG acyclic. This problem is formally defined as follows:

Definition 5: Weighted minimum feedback edge set problem: Given an edge-weighted directed graph $G(V, E)$ with cycles, the goal is to find a set of edges $A \subset E$ with the minimum total weight such that $G'(V, E - A)$ is acyclic.

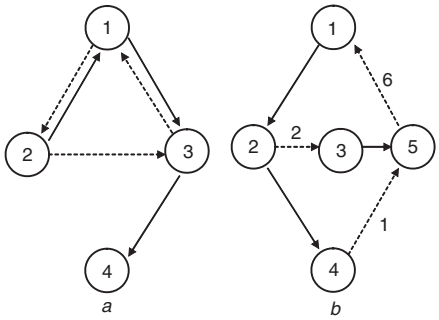

Fig. 7 *Removal of cycles*
a Crossing edges may create cycles
b BackEdge will break the two cycles in a single pass (by removing (n_5, n_1)), whereas cycleBreaker takes two passes: first to remove (n_2, n_3) and next to remove (n_4, n_5)

Since the non-weighted version of the minimum feedback edge set problem is NP-complete [22], it is not hard to see that this weighted version also becomes NP-complete.

We note that a simple heuristic exists to solve the weighted minimum feedback edge set problem: perform DFS once and remove all backward (and crossing) edges found. This will guarantee that the remaining VCG is acyclic. A major shortcoming of this approach named backEdge is that it ignores the edge weights and may produce a feedback edge set with a high total weight. An illustration is shown in Fig. 7*b*, where backEdge removes (n_5, n_1) and loses wire crossing reduction of six to break the two cycles. However, a better approach is to remove (n_2, n_3) and (n_4, n_5) to break the cycles and only lose wire crossing reduction of three. Note that this second approach named cycleBreaker requires us to enumerate all cycles in VCG. DFS, however, can not provide all cycles in a single pass as illustrated in Fig. 7*b*. DFS can either detect $(1, 2, 3, 5, 1)$ or $(1, 2, 3, 5, 1)$ but not both in a single pass. Therefore, it takes several iterations to remove all cycles with this cycle-by-cycle approach that searches for the minimum-weighted crossing edge.

We present our cycle breaker algorithm in what follows:

Cycle breaker algorithm
1: while (the VCG contains cycles)
2: perform depth-first traversal(VCG);
3: C = set of detected cycles in VCG;
4: while ($|C| > 0$)
5: for (each crossing edge $e \in C$)
6: setKey(e, C);
7: e = edge with the highest key in C;
8: remove e from VCG;
9: remove all cycles containing e from C;
setKey(e, C);
1: $key(e) = count = 0$;
2: for (each cycle $c \in C$ that contains e)
3: $count = count + 1$;
3: for (each edge $e' \in c \neq e$)
4: $key(e) = key(e) + w(e')$;
5: $key(e) = count \cdot key(e)/w(e)$;

As the cycleBreaker function traverses the VCG in depth-first order, it collects the cycles detected. When this traversal is complete, it will have added some cycles from each strongly connected subgraph in the VCG to the collection of cycles that must be broken. Because the VCG was acyclic until the crossing edges were added, each cycle must depend on the presence of a crossing edge. Once these cycles are found, the crossing edges of each are keyed according to the setKey function. We use the following function to determine the edge key value

$$key(e) = \frac{|c(e)| \cdot \sum_{e' \in c(e)} weight(e')}{weight(e)}$$

where $c(e)$ denotes the cycles that contain e. The edge with the highest key is removed from the VCG, and the cycles that contain this edge are removed from the collection of detected cycles. The edges are rekeyed and further cycles are removed until no more detected cycles remain unbroken. In this manner, small-weighted edges that occur in many cycles with other higher-weighted edges are more likely to be targeted for removal to break cycles. Note that these key values will change upon each edge/cycle removal. Thus, we use a priority heap to accommodate the dynamic key update. As discussed previously, cycleBreaker runs multiple iterations of 'DFS plus cycle removal' until all cycles are removed from the VCG.

4.4 Wire crossing against channel width trade-off

After our VCG becomes acyclic again, the last step is to assign each net to a unique routing track, i.e. to route each net. We note that the net-merging algorithm used in YK router [18] may not be a good option for wire crossing minimisation. YK attempts to merge nets without increasing the longest path from source to sink in the VCG since it is the theoretical track count lower bound for the channel. In addition, nets can only be merged if they are not on the same path in the VCG. Since we insert crossing edges to the VCG, the longest path length could increase, and thus merging opportunities could be reduced. Therefore, the task of the YK algorithm is much harder with the presence of these crossing edges than without. These more complicated vertical relationships between overlapping nets often require a larger number of tracks to contain them than a solution that does not attempt to minimise the resulting crossing count. Thus, the routing track assignment is done by the LEF [20] algorithm.

Consider Fig. 8, where the contact edges do not dictate a relationship between Nets 2 and 3. These nets can be routed using only two tracks, but more crossings can be removed by adding a crossing edge from Net 1 to Net 2. Again, this routes Net 1 between Nets 2 and 3, reducing the crossings, but, this time, increasing longest path and the track count. Since each net is on the same path, no nets can be merged. When the longest path in the VCG increases, the lower bound of the track count increases, often resulting in wider channels. Thus, channel routing solutions that reduce crossings tend to increase the track count of the circuit.

5 Experimental results

Our algorithms were implemented in C++/STL, compiled with gcc v2.96 run on Pentium III 746 MHz machine.

5.1 QCA duplication results

Twenty combinational and sequential circuits of various sizes were selected for our experiments. The bigx are industry circuits, bxx.opt from the ITC benchmark [23], and the remaining from the ISCAS benchmark [24]. The circuits were initially converted to directed graphs, involving the

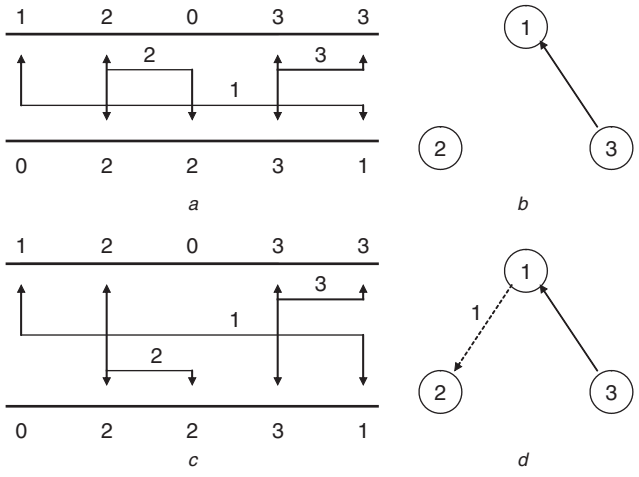

Fig. 8 *Overlap of Nets 2 and 3*
a, *b* A similar channel and VCG to Fig. 7
c Optimum solution for minimising wire crossing
d Corresponding modified VCG with an increased longest path

Table 2: QCA placement results for smaller circuits

	Barycenter			Barycenter w/SA		
ckt	x.ing	area	%x.r	x.ing	area	%x.r
Adder	409	1.00	0.0	219	1.00	46.5
k2	810 K	1.00	0.0	703 K	1.00	13.2
c1908	17.7 K	1.00	0.0	13.8 K	1.00	22.5
Pair	80.4 K	1.00	0.0	80.2 K	1.00	0.3
i8	588 K	1.00	0.0	584 K	1.00	0.6
c3540	73.2 K	1.00	0.0	58.9 K	1.00	19.5
t481	3.69 M	1.00	0.0	3.68 M	1.00	0.3
i10	300 K	1.00	0.0	280 K	1.00	6.6
s5378	114 K	1.00	0.0	85.9 K	1.00	24.4
c7552	163 K	1.00	0.0	105 K	1.00	35.5
s9234	368 K	1.00	0.0	260 K	1.00	29.1
Time		9 s			819 s	

removal of registers for the sequential circuits. Table 1 shows the result of KLBG construction. The number of layers (♯layer) indicate the length of the critical path. The number of bypass nodes inserted (♯bypass) in general may seem high at first, as they are 5–10 times larger than that of the original nodes. However, we need to remind ourselves that bypass nodes are just QCA wires and aid describing the connections between real nodes in a KLBG representation of the circuit. The benchmark circuits were divided into two sets according to their sizes, 8000 logic gates or nodes being the boundary. This was primarily due to simulated annealing; performing the process for larger circuits would require impractical computation times. Neighbour-merging was applied to all circuits as a post-process before the evaluation of comparison metrics.

Table 2 shows that simulated annealing improves upon the heuristics of Barycenter as expected. Observing the percentage of crossings removed (%x.r), up to 46.5% of wire crossings (compared to the result of Barycenter) were able to be removed without duplication costs. Table 3 shows the results of our FTD algorithms. For both FTDs, the tolerance was controlled so that the ratio of area (area), estimated with the number of nodes present, was as closely matched to 2.00. For priority FTD, even 0% tolerance (maximum area) could not double the area in many cases, indicating once again that the fan-out region is the minority region of the circuit. In general, 0% priority FTD removed around 60–70% of the total wire crossings with less than 50% increase in area. To measure the effectiveness of duplicating prioritised nodes, we compare its results to that of regular FTD with wire crossings reduced per duplication (x.r/d) to compare the two FTDs. Except for adder, the smallest circuit, priority FTD proves to have performed more effective or equal duplications. In addition, the difference in x.r/d seems to become more significant as the circuits get larger.

Table 4 shows the placement results for the larger circuits where Barycenter was followed by priority FTD with 0% tolerance. Data indicate that our placement algorithm allows the removal of about half the wire crossings in the circuit with a very small increase in area. In circuits such as big2 and big3, the area of the circuit rather decreased after duplication as neighbour-merging removed redundant nodes. As these circuits have more layers and columns, there is a distinctive prioritised region with possibly largely split fan-outs. Priority FTD taking advantage of this is clearly shown with the x.r/d measurement as it increases proportionally with the size of the circuit.

Table 3: QCA placement results for smaller circuits

	Priority FTD						Regular FTD					
ckt	x.ing	area	%x.r	♯dup	x.r/d	tol	x.ing	area	%x.r	♯dup	x.r/d	tol
Adder	54	1.05	86.8	33	10.8	0	107	1.03	73.8	21	14.4	13
k2	491 K	1.99	39.3	347	918.9	73	491 K	1.99	39.3	347	918.9	73
c1908	7835	1.20	55.8	1655	6.0	0	13.1 K	1.16	26.0	1262	3.7	39
pair	14.7 K	1.08	81.7	718	91.6	0	27.7 K	1.07	65.6	632	83.5	24
i8	279 K	2.01	52.5	8156	37.9	23	326 K	2.01	44.5	8203	31.9	29
c3540	27.6 K	1.16	62.3	1176	38.8	0	36.8 K	1.16	49.7	1179	30.9	30
t481	0	1.88	100.0	6301	586.5	0	0	1.88	100.0	6301	586.5	0
i10	104 K	1.43	65.2	12.5K	15.6	0	261 K	1.52	13.0	15.0 K	2.6	32
s5378	37.3 K	1.30	67.2	3740	20.4	0	53.3 K	1.30	53.0	3748	16.1	28
c7552	72.7 K	1.08	55.5	1532	59.3	0	102 K	2.78	37.3	27.7 K	2.2	19
s9234	108 K	1.25	70.5	9014	28.7	0	202 K	2.38	45.1	49.4 K	3.4	30
Time			18 s						20 s			

Table 4: QCA placement results for larger circuits (FTD tolerance was set to 0% for all circuits)

	Barycenter			Priority FTD				
ckt	x.ing	area	%x.r	x.ing	area	%x.r	♯dup	x.r/d
b15_opt	2.06 M	1.00	0.0	706 K	1.58	65.8	54.4 K	24.9
s13207	469 K	1.00	0.0	211 K	1.05	55.0	3919	65.8
s15850	394 K	1.00	0.0	161 K	1.02	59.1	4054	57.5
b21_opt	3.06 M	1.00	0.0	2.60 M	1.01	14.9	2921	156.2
B17_opt	8.62 M	1.00	0.0	2.51 M	1.28	70.9	86.1 K	71.0
big3	36.4 M	1.00	0.0	32.2 M	0.98	11.6	3752	1123.4
big1	21.0 M	1.00	0.0	9.74 M	1.01	53.6	8225	1368.0
big4	22.0 M	1.00	0.0	12.1 M	1.00	44.8	7601	1296.4
big2	82.9 M	1.00	0.0	59.0 M	0.96	28.9	11.2 K	2139.9
Time		2296 s				2853 s		

Table 5: QCA channel routing results

	LEF		YK		BE		CB	
ckt	xing	track	xing	track	xing	track	xing	track
cmb	0	42	0	42	0	42	0	42
decod	26	65	26	65	26	65	0	77
Pm1	56	28	52	28	50	29	1	43
i1	126	113	112	110	97	119	17	136
sct	2447	82	2180	79	2025	99	73	140
adder	0	72	0	72	0	72	0	72
i2	34 30	131	3388	142	3319	158	2270	281
x4	58 672	372	54 528	402	50 656	523	2242	1253
k2	533 694	858	493 600	874	375 464	1274	92 059	1692
i5	28 443	596	27 260	624	25 602	669	190	1317
apex6	64 361	739	62 877	807	58 498	902	605	2755
rot	150 372	1011	143 856	1108	144 636	1163	3289	4133
Frg2	666 234	1358	630 384	1614	620 448	1878	7219	7128
pair	554 124	2348	530 954	2643	527 186	2850	6599	12516
runtime	6857		7279		8080		7367	

5.2 QCA channel routing results

Table 5 shows the comparison of our cycleBreaker (CB) algorithm to the LEF algorithm (LEF), and Yoshimura and Kuh (YK) [18]. It was also compared to the backEdge algorithm (BE) discussed in Section 3 (a topological feedback edge set implementation). We note that LEF, YK, and BE generated channel routing solutions well above the circuits theoretical minimum crossing (x-low), but very close to the theoretical minimum track count (t-low). Our channel routing method CB comes much closer to the theoretical minimum crossing count, but produces wider channels. Note that our channel routing method completes in only 7.5% more runtime than the simplest alternative method: LEF. Not only does our algorithm very closely approach the theoretical lower bound of wire crossing for each circuit, but it does so in a time-efficient manner.

Table 6 first shows the ratio of the resulting wire crossing count from each method to the theoretical lower bound of wire crossing for each circuit. The average LEF result has wire crossing 19x above the lower bound. The average result by the YK method is 18x above the lower bound, and the average BE result is almost 17x above the lower bound. In comparison, the method presented here results in average wire crossing only 13% above the lower bound, providing superior wire crossing minimisation. Table 6 also shows how the wire crossing count and track count are inversely proportional. While our channel routing method produces wire crossing much closer to the theoretical lower bound, the resulting channel width is almost 6x greater than the theoretical lower bound. The LEF, YK, and BE methods produce channels about 40–80% greater than the lower bound.

Table 6: Comparison of wire crossing and track usage values to their theoretical lower bounds

ckt	LEF	YK	BE	CB
xing	20.37	19.26	17.86	1.13
track	1.43	1.58	1.80	5.78

6 Conclusions

In this article, we have presented new algorithms for wire crossing reduction during placement and channel routing. The FTD algorithm reduces wire crossing by node duplication. With its tolerance parameter, the user is able

to select a trade-off point between the minimum number of crossings with no area cost and the minimum area needed to guarantee zero wire crossing. Our folding post process reshapes the circuit to a square-like shape, reducing wire skew and possibly the overall chip area. Our channel routing algorithm reduces the crossing count to very near the theoretical lower bound, as it targets the absolute minimum number of wire crossings to route a particular circuit. We also identified a tradeoff between wire crossing and channel width in QCA channel routing problem.

7 Acknowledgment

This research is partially supported by the National Science Foundation under NER-0404011.

8 References

1 Amlani, I., Orlov, A., Snider, G., and Lent, C.: 'Demonstation of a func. quantum-dot cellular automata cell', *J. Vac. Sci. Technol.*, 1998, pp. 3795–3799
2 Lieberman, M., Chellamma, S., Varughese, B., Wang, Y., Lent, C., Bernstein, G., Snider, G., and Peiris, F.: 'Quantum-dot cellular automata at a molecular scale', *Ann. New York Acad. Sci.*, 2002, pp. 225–239
3 Snider, G., Orlov, A., Amlani, I., Bernstein, G., Lent, C., Merz, J., and Porod, W.: 'Quantum-dot cellular automata: Line and majority gate logic', *Jpn. J. Appl. Phys.*, 1999, pp. 7227–7229
4 Amlani, I., Orlov, A., Toth, G., Bernstein, G., Lent, C., and Snider, G.: 'Digital logic gate using quantum-dot cellular automata', *Science*, 1999, **284**, pp. 289–291
5 Kummamuru, R., Timler, J., Toth, G., Lent, C., Ramasubramaniam, R., Orlov, A., and Bernstein, G.: 'Power gain in a quantum-dot cellular automata latch', *Appl. Phys. Lett.*, 2002, **81**, pp. 1332–1334
6 Walus, K., Dysart, T., Jullien, G., and Budiman, R.: 'QCADesigner: a rapid design and simulation tool for quantum-dot cellular automata', *IEEE Trans. Nanotechnol.*, 2004, **3**, pp. 26–31
7 Antonelli, D., Chen, D., Dysart, T., Hu, X., Kahng, A., Kogge, P., Murphy, R., and Niemier, M.: 'Quantum-dot cellular automata partitioning: Problem modeling and solutions'. Proc. ACM Design Automation Conf., 2004
8 Lim, S.K., Ravichandran, R., and Niemier, M.: 'Partitioning and placement for buildable QCA circuits', *ACM J. Emerging Technol. Comput. Syst.*, 2005, **1**, pp. 50–70
9 Tougaw, P., and Lent, C.: 'Logical devices implemented using quantum cellular automata', *J. Appl. Phys.*, 1994, **75**, pp. 1818–1825
10 Garey, M., and Johnson, D.: 'Crossing number is NP-complete', *SIAM J. Algebr. Discrete Methods*, 1983
11 Weiskircher, R., Gutwenger, C., and Mutzel, P.: 'Inserting an edge into a planar graph'. ACM-SIAM Symp. on Discrete algorithms, 2001
12 Sugiyama, K., Tagawa, S., and Toda, M.: 'Methods for visual understanding of hierarchical system structures', *IEEE Trans. Syst. Man. Cybern.*, 1981, **SMC-11**, pp. 109–125
13 Catarci, T.: 'The assignment heuristic for crossing reduction', *IEEE Trans. Syst. Man Cybern.*, 1995, **25**, pp. 515–521
14 Marek-Sadowska, M., and Sarrafzadeh, M.: 'The crossing distribution problem', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1995, **14**, pp. 423–433
15 Enos, M., Hauck, S., and Sarrafzadeh, M.: 'Evaluation and optimisation of replication algorithms for logic bipartitioning', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1999, pp. 1237–1248
16 Rkic, M., Lillis, J., and Beraudo, G.: 'An approach to placement-coupled logic replication'. Proc. ACM Design Automation Conf., 2004
17 Beraudo, G., and Lillis, J.: 'Timing optimization of FPGA placements by logic replication'. In Proc. ACM Design Automation Conf., 2003
18 Yoshimura, T., and Kuh, E.: 'Efficient algorithms for channel routing', *IEEE Trans. Comput.-Aided Des. Int. Circuits Syst.*, 1982, **CAD-1**, pp. 25–35
19 Sapatnekar, S.: 'A timing model incorporating the effect of crosstalk on delay and its application to optimal channel routing', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2000, **19**, pp. 550–559
20 Hashimoto, A., and Stevens, S.: 'Wire routing by optimizing channel assignment within large apertures'. Proc. ACM Design Automation Conf., 1971, pp. 155–169
21 Deutch, D.: 'A dogleg channel router'. Proc. ACM Design Automation Conf., 1976
22 Garey, M.R., and Johnson, D.S.: 'Computers and intractability: a guide to the theory of NP-completeness' (Freeman, San Francisco, 1979), pp. 209–210
23 ITC99. The ITC 1999 benchmark suite
24 ISCAS89. The ISCAS 1989 benchmark suite