ORCA A Sea-of-gates Place and Route System

Mitsuru Igusa, Mark Beardslee and Alberto Sangiovanni-Vincentelli

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, California 94720

Abstract

Sea-of-gates is becoming an important design style for Application Specific Integrated Circuits (ASICs). The sea-of-gates technology offers more flexible placement and routing options not available in gate arrays. Very few systems are available today that can automatically layout sea-of-gates and none of these systems effectively use the features available in sea-of-gates architecture. ORCA is a place and route system for sea-of-gates, whose objective is to produce the highest density layout by fully exploiting the inherent features of this new design style. The ORCA system starts with a module generator which preprocesses memory arrays and other logic with a regular structure to form high density macros. The remaining logic is clustered together to form flexible macros, which we call *porous*. The porous macro-cells allow global routing to pass through the macro instead of detouring around its perimeter. The porous macros are dynamically shaped and resized by interaction with global wiring analysis. Finally, a general channelless area router has been developed to address the multiple layers of interconnect and routing areas which will be dominantly over-the-cell. Due to the large size of the problem (e.g. 100,000 gates), the placement and routing algorithms are hierarchical.

1 Introduction

The "gate–array" (also referred to as master–slice, or uncommitted logic array) is by far the most common design style for ASICs. The computer aids for gate–array design are also the most advanced and the most mature. In this approach, a two–dimensional array of replicated transistors is fabricated to a point just prior to the interconnection levels. Generally a two–level interconnection scheme is used for signals and, in some approaches, a third more coarsely defined layer of interconnections is provided for power and ground connections. Because one or more interconnection layers are used within a group of transistors to define the function of a cell, these intra–cell interconnections often block the passage of more global inter–cell connections. For that reason, and to simplify the placement and routing problems associated with these arrays, the inter–cell connections are implemented on a rectilinear grid in the *"channels"* between the cells. Over-the-cell interconnections are typically characterized by straight 2nd layer metal *"feedthroughs"*.

This recent variation of the gate–array structure is often referred to as a *"sea–of–gates"* array [1] and, as well as providing new challenges for the CAD community, promises to replace many of the designs previously undertaken in the conventional gate–array style. The routing area is not organized *a priori* into channels as in standard gate–arrays. The basic building blocks in this design style are similar to the ones used in standard gate–arrays. In the sea–of–transistor architecture, using CMOS processing technology, transistors are in a compact regular arrangement in rows (columns) of complementary pairs.

Given the tight arrangement of the transistors, it is possible to design large macros effectively with little area penalty. For example, it is possible to design RAM blocks using the transistor array. This capability makes the sea–of–gates approach more competitive than gate-array and possibly an alternative to some low volume standard-cells and macro-cells. However, the automatic generation of chips which fully utilize these inherent capabilities of sea-of-gates is more difficult than with the standard gate–array design style.

Three layers of interconnects appear more often in the larger VLSI designs in this design style due to the potential increase in chip utilization and performance (i.e. shorter wire length and decreased capacitance). The active area utilization could be dramatically increased, but the wiring problem becomes more complex and must efficiently deal with over-the-cell routing.

In this paper, we present the ORCA system, which consists of module generators, placement, and routing tools. The system is designed to support a variety of layout styles:

1. standard–cell like, where the basic cells are arranged in rows of equal height and the routing area

is arranged into channels with flexible dimensions;

2. macro–cell like, where macros can be built using basic cells belonging to different rows of active devices, and macros as well as cells can occupy any position of the chip within the constraints of the architecture;

3. porous macro–cell, where the basic cells are clustered together to form macros which are flexible in shape and allow routing to pass through the macro. The key feature of this method is that it allows routing to go straight through macro blocks instead of detouring around, thus reducing wiring length and consequently the chip area (Figure 1).

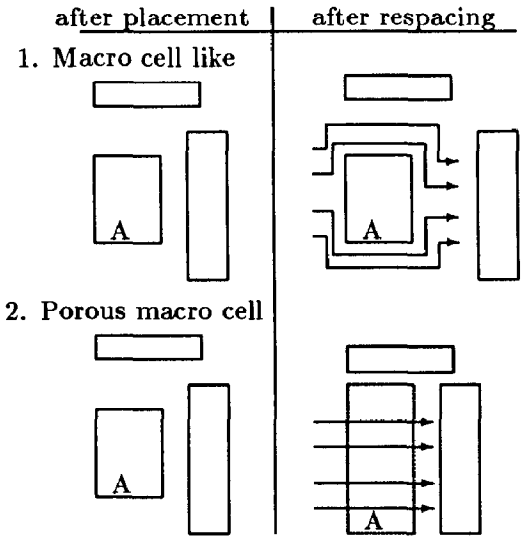

Figure 1: Porous macro-cell

2 Overview of the ORCA algorithm

The ORCA system consists of a floorplanner, a placer, a global router, a spacing requester, a macro adjuster and an area router. The *floorplanner* partitions the logic circuits into groups of cells and assigns the cells to areas of the layout. The *placer* places the logic circuits within each layout area. The *global router* performs a "coarse" routing of the chip and locates the routing areas that are too congested. The *spacing requester* decides which areas of the layout need to be expanded or contracted based on the global routing congestion information. The *placement adjuster* readjusts the location and placement of cell groups to accommodate the requested spacing changes. The global routing and placement improvement sequence is continued until it is determined that the layout can be routed. Finally, the *area router* does the detailed routing.

3 Floorplanning and Placement

The size of a sea–of–gates layout can be as large as 100,000 gates, and the largest available sea–of–gates arrays will undoubtedly become even bigger in the future. While other steps of the sea–of–gates physical design system may subdivide their problem into a smaller more managable subtasks, the floorplanning/placement step must, at some point, deal with the entire design all at the same time. To be able to handle such large circuits, fast and memory efficient algorithms are used.

One algorithm that is effective and fast is the min–cut algorithm [2]. A variation of this algorithm [3] has been shown to complete in time linearly proportional to the size of the network. The min–cut algorithm takes a general network and divides it into two groups so that the number of nets that connect cells in different groups is minimized and the two groups have approximately the same size. This has the effect of grouping cells that are highly connected so that they are close together, thereby minimizing the amount of wiring in the final layout.

The floorplanning step performs assignment of all the cells to restricted areas of the layout space by using the min–cut algorithm to hierarchically partition the circuit. The process begins by first making a cut in the network. Then two groups of cells resulting from the cut are constrained to lie inside respective halves of the layout space. The two resulting pieces will then be partitioned with a cut line perpendicular to the first. The floorplanning hierarchy can now be represented as a tree with the leaf nodes being groups of cells and the internal nodes representing network cuts. The cuts are continued recursively with the cut directions changing at every level of the tree until the groups of cells at the leaves are small.

When partitioning at lower levels of the tree, one should consider the effects of the locations of the neighboring cells on the cut being done. In our method, all cuts that reside at the same level of the tree with the same cut direction and on the same horizontal or vertical line are done simultaneously. At each step the cut line is defined so that it cuts a sequence of groups that extend entirely from one side of the layout to the other. Figure 2 illustrates the idea; the horizontal cuts a, b, c and d are all considered simultaneously. We found that this method is better than considering partitions sequentially with a pin propagation scheme [4].

It has also been noticed that the quality of a cut depends on the parts of the layout whose locations have already been determined. The cells and i/o pads that have already been placed are allowed to exert an influence on the cut being performed. At every cut the area of the two groups is balanced, thus at the end of

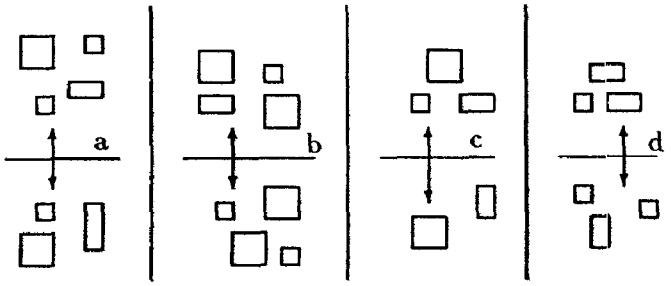

Figure 2: Simultaneous min-cut

the floorplanning stage all of the groups should have approximately the same area.

Since the min–cut algorithm is a heuristic based deterministic algorithm, the result of the algorithm is very dependent on the initial starting conditions. A simple yet effective method to fix this problem is to perform the min-cut algorithm on the same cut many times; each time starting from a different random initial state. When using many random initial starts, the measured routing length has been observed to decrease from 10 to 15 per cent.

At this point the relative locations of the cell groups are determined but their exact locations and shapes are not. To determine the shape that a particular cell group should take, we first generate all possible shapes that it can take then choose one. The possible shapes are generated by first trying to pack all the cells into a shape one row high and as wide as necessary. Then we try to pack it into two rows, and on up until it forms a shape that is only one column wide and as many rows as necessary. Using the array of shapes generated for each group and a binary tree that represents the sequence of cuts made in the floorplanning stage, the locations and shapes of the groups is then determined using a "shape adding" approach [5]. The algorithm chooses the shape for each group that minimizes the overall chip area and produces a chip with a desired aspect ratio or a chip with an aspect ratio in a predefined range.

An optimization that can be performed at this stage is to introduce flexibility into the directions of the cuts. In situations illustrated in Figure 3, the overall layout area is improved when the cut directions are changed (Figure 3b). At each appropriate position in the partition tree, both cut directions are tried and the results incorporated into the shape adding process. Once the shape adder has determined the shape of the layout, the best direction of all the cuts in the partition tree can be determined.

Once each group's shape is determined, its cells are placed in a locally optimum configuration by exhaustive enumeration. The objective of the group placement is to minimize total wire length. This step is repeated since it is dependent on the order in which the groups

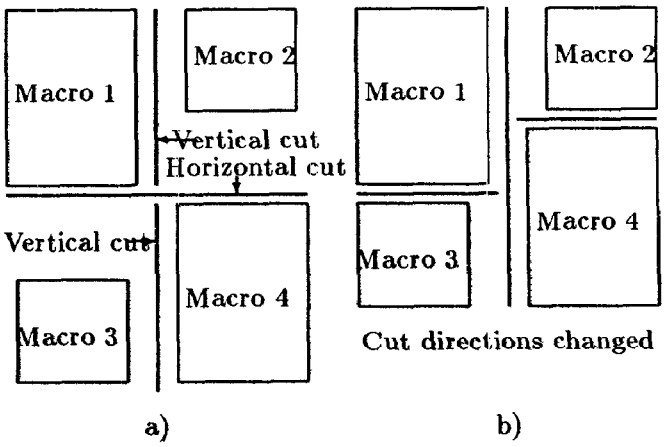

Figure 3: Alternate cut directions for improved placement

are processed. In practice, after two or three iterations the total wire length does not improve further.

To deal with timing constraints, the user can specify upper bounds on the lengths of critical nets. At each stage of the mincut hierarchy, the placement program estimates the length of critical nets and adapts its placement procedure to meet the upper bound constraints. When the estimated length of a critical net approaches its upper bound, the mincut algorithm attempts to assign cells on the critical net to a side of the mincut partition, such that the upper bound constraint is maintained.

4 The routing area estimation and placement adjustment

The floorplanning/placement step does not directly take into account the space needed for wiring. In order to ensure that the placement produced can be successfully routed, we estimate the routing requirements using a simplified, fast but less accurate version of the global router used in the area router (discussed in Section 5). The global router determines the amount of additional space required in the areas which are too congested (where the estimated routing usage exceeds the routing capacity). The placement adjustment phase then incorporates these requests for space into the layout. The relative placement defined by the cut tree is not altered; instead, the extra space needed is incorporated into the appropriate shape function of the cell group in that locality, and then the "shape adding" phase is preformed again. The placement adjustment is very fast since the circuit need not be repartitioned.

The global router and placement adjustment programs iterate until convergence to a design that can be successfully routed at the detailed level. The inter-

action of the global router and the placer makes the layout appear like a large "porous" structure. This is because area is added in regions of high routing density, and as a result the wires are allowed to form paths that minimize their length and need not be detoured due to routing congestion.

5 Area Router

The general purpose ORCA area router is intended for a wide variety of routing problems which includes sea-of-gates, macro-cell and gate-array routing. A powerful feature of this router is that it is a general channelless router that handles over-the-cell routing and routing of 2, 3, or more layers of interconnect. The router is given a set of terminals to interconnect and a description of the blockages, which can be of any shape and on any routing layer. The area router assumes a preferred routing direction for each routing layer.

The area router is based on the well known shortest path algorithm commonly called the mazerouter [6]. The shortest path algorithm is used for computing paths in the global routing, detailed routing, and intermediate routing phases. The global routing algorithm selected was that used at IBM [7] with further enhancements suggested by Nair [8]. The global routing algorithm uses a shortest path algorithm which penalizes paths crossing heavily congested areas, and uses rip-up-and-reroute to reduce order dependency of the initial routing. The global routing results were favorable. In the ORCA router, this idea of using a mazerouter combined with rip-up-and-reroute has been extended down into detailed routing by successively refining the initial global routing grid into finer and finer grids until each global routing cell becomes a detailed routing grid as in [9]. Thus, we have one simple general algorithm that is used for both global routing and detailed routing.

The area routing algorithm uses the classical global routing model. The routing area is physically divided by a set of horizontal and vertical cut lines into a rectangular matrix of routing tiles. The cut lines are positioned such that they coincide with channel boundaries [1]. Each routing tile is a face in the graph $G = (V, E)$. The routing algorithm uses the dual graph $G^\star = (V^\star, E^\star)$. Each edge $e \in E^\star$ has the following properties: *length*, routing *capacity*, routing *demand*, and *expandability*. The *length* of an edge $e \in E^\star$ is the center-to-center distance of the tiles corresponding to the endpoints of e. The routing *capacity* of an edge of E is maximum number of feasible routes (on all layers of the preferred direction) that can cross the corresponding edge in the dual graph E. The edge capacity is computed by counting the number of routing tracks that span the midpoint of each tile adjacent to the edge. The *expandability* property is used to identify edges that have variable routing *capacity* which designate variable width channels.

A routed *path* of a net n, P_n is a tree in $G^\star$ that connects the terminals of the net. A routing *subpath* $P \subset P_n$ is a connected path whose endpoints in P_n are either terminals or vertices with more than 2 edges.

Given the routing model, the objective of the router is to find routing paths for each net such that no edge $e \in E^\star$ has $demand(e) > capacity(e)$.

Approximations to Steiner trees are used for initial global routing to minimize total wire length. A greedy method [10] is used to compute the approximation to the Steiner tree. The initial global routing is preformed without regard to congestion; only interconnection length is minimized. The next phase of global routing addresses the routing congestion.

A rip-up-and-reroute algorithm is used to move routing from highly congested routing areas. Highly congested areas are identified by the edges in the global routing graph which have routing demands that exceed their capacities. The rip-up-and-reroute algorithm selects the most congested edge in the global wiring graph, randomly selects a global wire subpath which traverses the edge, and deletes the wire path from the database, and then tries to find a better wiring path using a shortest path algorithm. The mazerouter or minimal cost path search algorithm [6] is directed by a cost function which assigns high cost to heavily congested edges. The algorithm is greedy. Only wiring paths with lower wiring costs are accepted. The rip-up-and-rerouting is continued until none of the capacities of each global wiring edge are saturated.

The cost function used in the shortest path algorithm is linear, i.e. the cost of a path is the sum of the cost of each component (or edge) in the path. The edge cost is a function of its length penalized when the edge is over saturated. In mathematical terms, the objective of the path search algorithm is,

$$\min_{paths\ in\ G^\star} cost(path) = \min \sum_{e \in path} edgecost(e),$$

$$edgecost(e) = length(e) * f(overflow(e), expandability(e)),$$

$$overflow(e) = demand(e) - capacity(e).$$

$f(overflow(e), expandability(e))$ is the penalty function for overflowed edges ($overflow(e) > 0$) that also takes into account the expandability of edge e.

The global wiring model is "refined" by subdividing each global routing row and column into smaller subrows and subcolumns. Figure 4 shows the added cut lines (shown in dotted lines) in the refined global routing graph. At each iteration of global routing, the refinement of the global wiring model will give a more

[1] In cases where there are no channels, the cut lines are positioned as close as possible to edges of the boundaries of instances of circuits used in the design.

exact picture of where the net paths should go. At the very last stage of refinement, each global routing tile would correspond to the intersection of one vertical and one horizontal wiring track. The nets are then routed in the refined global wiring model using the shortest path search algorithm, confined to the path suggested to the higher level global route, without regard to congestion.

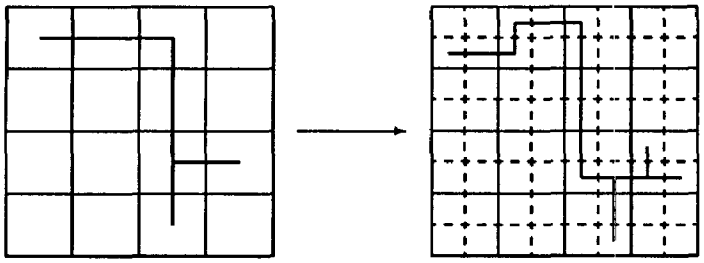

Figure 4: Ripup and reroute path search windowing

To reduce the complexity of the shortest path search algorithm, the higher level global routing paths are used as a guide to limit the domain of the lower level path search algorithm. The path search is limited by the wider path suggested by the higher level routing. This reduces the complexity of the path search algorithm to $O(length\ of\ path)$ instead of $O((length\ of\ path)^2)$. Figure 4 shows how a higher level global wire path is used as a guide in the refined global routing graph.

In the event that overflows still exist, further iterations of rip-up-and-reroute are performed on an enlarged window extending past the path suggested by the higher level routing. An extra component is added to the cost function to guide the mazerouter to prefer the path suggested by the higher level routing.

When the global wiring model is refined to the point where it matches the detailed routing model, the edge cost function is changed to influence the mazerouter in chosing vias, wrong-way-wires, and overlaps. As in the coarse routing model, an overflow in the detailed routing phase is present when the capacity along an edge exceeds the demand. However in the detailed routing model, the capacity along an edge is either one or zero, and when an overflow occurs this means a wire going through a blocked region (i.e. $capacity = 0$ and $demand >= 1$) or a wire shorting another wire (i.e. $capacity = 1$ and $demand >= 2$). The cost function used in the detailed routing phase distinguishes between these two cases, and prefers the latter. Also there are two types of overflows, "crossovers" and "overlaps". The crossovers occur when a net shorts another net by going from one side of the shorted net to the other. An overlap is defined when a net shorts another net but is not a crossover. By experimental analysis, we found that when we allow overlaps rather than crossovers, the number of overflows after rip-up-and-reroute is substantially reduced. The cost function penalizes, in order of highest-to-lowest severity, blockages, crossovers, overlaps, wrongway wires, and finally vias.

Even though the complexity of the algorithm is kept low by the windowing scheme, the memory requirements grow geometrically with each refinement of the global wiring graph. The solution to the memory problem is to subdivide the routing problem into independent subproblems. The routing graph G is partitioned at user specified horizontal and vertical cut lines (Figure 5). The problem then becomes the assignment of routing paths which cross the partition boundaries. New pseudo terminals are assigned to locations along the boundaries for these routing paths. Each pseudo terminal along one edge $e \in E \cap boundary\ edges$ is given a $termAssignCost$ for each possible location along the edge, where the most "desired" location of a terminal is given the lowest $termAssignCost$. Nets which have terminals very near edge e are given the highest priority $termAssignCost$. The assignment algorithm assigns each pseudo terminal to a location which minimizes the total sum of each terminal/location cost.

The assignment of terminals along partition boundaries introduce extra artifical constraints that degrade the detailed routing process. By experiment, we found that along these boundaries, we had many wiring violations after detailed wiring. To solve this problem, we defined new partition boundaries that are offset from the original boundares, merged the routing across the original boundaries and rerouted wires along the merged boundaries.

To handle three and more layers of routing, the area router assigns routing subpaths to HV layer pairs just before the global routing model is refined to the last most detailed stage. The global routing model is duplicated for each layer, and the capacities of each edge are updated to reflect only the routing tracks found on its layer. Nets are routed one at a time in the multi-layer routing model in a first come first serve manner: the lowest HV layer pair is filled first, then the next HV layer pair, and so on.

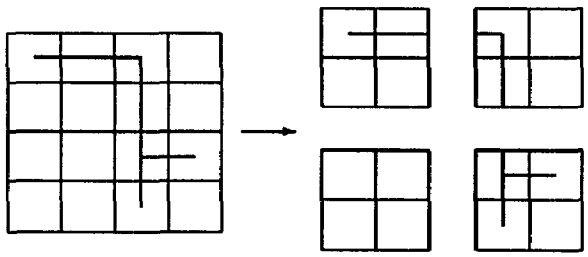

Figure 5: Hierarchical decomposition

6 Results

The ORCA system has been implemented in the C under the UNIX operating system. It is integrated with the Berkeley CAD Design Environment [11]. The program was run on a variety of examples shown in Table 1. The randomly generated random1 example was the most difficult to route due to high connections/cell ratio. The mcnc1 and mcnc2 test cases are the gate array benchmark examples from the 1988 International Placement-and-Routing Workshop at the Microelectronics Center at North Carolina (MCNC). The hughes1 and hughes2 examples are from industrial sea-of-gates designs. The successful routing of the hughes2 example, with close to 20,000 cells and over 20,000 nets, demonstrates the feasibility of the area router to handle large problems. The smaller examples were routed with 3 levels of global routing refinement hierarchies, and the large hughes2 was routed with 4.

Direct comparisons with other sea-of-gates systems could not be made for any of the examples. However, we could make comparisons of different systems on the standard gate array benchmark examples mcnc1 and mcnc2 from the MCNC workshop. Table 2 compares different place and route systems by the total net length of the routing of the mcnc2 example [2]. Even though there is a slight variation in the performance (measured by total net length) of the place and route systems, all the systems produced the desired result: a fully routed gate array chip. This comparison demonstrates that ORCA can do a reasonable job on standard row-based gate arrays as well as handle sea-of-gates.

name	*#nets*	*#cells*	*#layers*	*active area*	*cpu time* [3]
adder32	478	414	2	64%	1
random1	500	500	2	32%	14
mcnc1	904	752	2	43%	15
mcnc2	3029	3014	2	36%	47
hughes1	1645	1266	2	56%	19
hughes1	1645	1266	3	95%	21
hughes2	21727	19532	2	48%	547

Table 1: Routing results

7 Acknowledgements

The authors would like to express thanks to Alan Kramer, Gregory Sorkin and Amit Sharma for their contributions to the ORCA system. The authors would also like to thank Chi Ping Hsu from Hughes and Wayne Christopher for providing some of the sea-of-gates examples.

[2] The results of the mcnc1 gate array example were omitted since it was much easier to route than the larger mcnc2 example.

[3] VAX 8650 cpu minutes

name	*total net length*
ORCA	5.304
LTX2	5.496
GE	5.215
Proud	5.178

Table 2: Results on mcnc2 gate array example

References

[1] A. Hui et al. A 4.1k gates double metal hcmos sea of gates array. *Proc. IEEE CICC*, :15–17, May 1985.

[2] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, Feb 1970.

[3] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. *Proc. 19th Design Automation Conference*, 1982.

[4] A. E. Dunlop and B. W. Kernighan, A procedure for Placement of Standard VLSI Circuits. *IEEE Trans. on Computer-Aided Design*, CAD-4:92–98, 1985.

[5] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 57(3):91–101, June 1983.

[6] C. Lee. An algorithm for path connection and its applications. *IRE Trans. Electronic Computers*, EC-10:346–365, 1961.

[7] K. H. Khokhani N. Nan K. A. Chen, M. Feuer and S. Schmidt. The chip layout problem: an automatic wiring procedure. *Proc. 14th Design Automation Comf.*, :298–302, 1977.

[8] Ravi Nair. A simple yet effective technique for global wiring. *IEEE Trans. on Computer-Aided Design*, CAD-6:165–172, 1987.

[9] M. Burstein and R. Pelavin. Hierarchical wire routing. *IEEE Trans. on Computer-Aided Design*, CAD-2:223–234, 1983.

[10] Jr. J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.*, 7:48–50, 1956.

[11] D. Harrison et al. Data management and graphics editing in the Berkeley design environment. *Proc. ICCAD*, Nov 1986.