AUTOMATIC BUILDING OF GRAPHS FOR RECTANGULAR DUALISATION

Marwan A. Jabri

Laboratory for Imaging Science and Engineering, School of Electrical Engineering
University of Sydney, New South Wales 2006, Australia

ABSTRACT

Rectangular dualisation is a technique used to generate rectangular topologies for use in top-down floorplanning of integrated circuits. This paper presents an efficient algorithm that transforms an arbitrary graph, representing a custom integrated circuit, into one suitable for rectangular dualisation. The algorithm makes use of efficient techniques in graph processing such as planar embedding and introduces a novel procedure to transform a tree of biconnected sub-graphs into a block neighbourhood graph that is a path.

INTRODUCTION

A top-down design methodology enables designers of custom integrated circuits to plan and solve, at the early stages of the design process, sub-circuit intercommunication (block interconnection). Such a methodology permits the use of the "river routing" scheme, where only adjacent blocks may communicate. The process of solving the communication problem is performed possibly without knowledge of sub-circuit layout detail, and represents an important phase of a floorplanning process which aims at determining surrounds, shapes and final area of the blocks.
This paper presents an algorithm that has been developed as part of a program aimed at the automation of the top-down floorplanning process and described elsewhere [7,6,4]. We use connectivity graphs to represent abstracted views of integrated circuits. A node corresponds to a block, and an edge represents block interconnections. In common with other researchers [2,10,7], we divide the determination of block surrounds into two processes:

Process 1: Derivation of an adjacency graph from the functional block diagram of the circuit (see Figure 1),

Process 2: Generation of the possible rectangular topologies from the adjacency graph (see Figure 2).

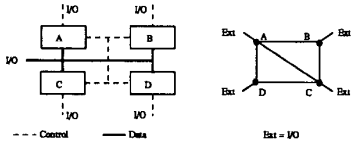

Figure 1: (a) A connectivity graph in the form of "boxes and buses" A, B, C and D represent sub-circuits or blocks. (b) An adjacency graph derived from (a). The existence of an edge between two blocks means that these blocks are adjacent and share a side.

In an adjacency graph, if two nodes share an edge, the blocks they represent share a side in the floorplan.
Graph rectangular dualisation [2,9,8,1] has developed as an efficient technique for the automation of Process 2. The technique transforms a graph representing a circuit into another graph called its Rectangular Dual (RD), where a node and an edge in the original graph correspond to a rectangle and adjacent rectangles (sharing a side) respectively in the RD. Figure 2 shows two rectangular duals of the graph of Figure 1(a). Not all graphs admit RDs. We call an adjacency graph that admits at least one RD a rectangular admissible connectivity graph (RACG). In particular, most integrated circuit connectivity graphs need a corresponding Process 1 transformation prior to rectangular dualisation.

Figure 2: These two floorplans represent two distinct rectangular topologies. Note the variations in the block surrounds.

Although the conditions of such a transformation have been reported [8], previous work on automatic floorplanning is still at the stage where a manual approach is used to implement Process 1 and to transform a graph into an RACG, as reported by Heller and associates [2]. Graph clustering is used for problem simplification, then manual transformations which include node additions are applied. Added nodes correspond to wiring blocks that solve the problem of communication crossings. But with circuit complexity increasing, human designers are overpowered by the combinatorial nature of this problem. This gives rise to the need for automation.
We present in this paper an algorithm that efficiently automates Process 1 by building an RACG that represents the circuit being floorplanned.

RACG PROPERTIES

Suppose that a graph G(N,E) represents the connectivity graph (or functional block diagram, FBD) of a circuit. As mentioned earlier, the properties of an RACG are formally presented by Kozminski and Kinen [8], and are summarised in the following definition and theorem.

Definition 1 *A block is a biconnected component. A plane block is a planar block. A shortcut in a plane block G is an edge that is incident to two vertices on the outermost cycle of G that is not part of this cycle. A Corner Implying Path (CIP) in a plane block G is a segment $v_1, v_2, ..., v_k$ of the outermost cycle of G with the property that (v_1, v_k) is a shortcut and that $v_2, ..., v_{k-1}$ are not endpoints of any shortcut. The Block Neighbourhood Graph (BNG) of a planar graph G, is a graph in which vertices represent the biconnected components of G, and where an edge between two vertices in the BNG exists if the corresponding biconnected components have a vertex in common. A Critical Corner Implying Path (CCIP) in a biconnected component G_i of G is a CIP of G_i that does not contain cut vertices of G.*

Theorem 1 *A planar graph G admits a rectangular dual iff:*

1. *Every face, except the external, has a degree of 3 (triangular).*
2. *All internal nodes have degree ≥ 4.*
3. *All cycles that are not faces have length ≥ 4.*
4. *One of the following is true:*

(a) G is biconnected and has no more than four CIPs.

(b) G has n, n ≥ 2, biconnected components; the BNG of G is a path; the biconnected components that correspond to the ends of this path have at most two CCIPs; and no other biconnected component contains CCIP.

THE RACG BUILDING ALGORITHM

Prior to algorithm presentation a theorem and some additional definitions are presented in the following paragraphs.
The following theorem makes the automation of condition 4 of Theorem 1 considerably efficient.

Theorem 2 *Consider a planar connected graph G(V,E) embedded on the plane, and consider an additional node present in the external face of the embedding and connected to each node at the periphery of the embedding defining thus a set of faces F. The BNG of G is a path iff F does not contain a face of degree > 3.*

Thus to force a BNG path for a planar embedding we ensure that the faces in F are triangular.

The Algorithm's Input

Consider a connected graph G(V,E) that represents the FBD of a circuit. The user can select the preferred set of blocks to occupy corners in the rectangular topology. However, corner selection has the lowest priority in the process, as in a top-down design strategy block communication port locations are not yet defined. The user can also select weights for the edges. These weights are used in a cost evaluation function in cases of competing solutions and to determine a critical path to be preserved during the planar embedding process.
In order to represent the external connections (I/O pads), an additional node is introduced. This node will be referred to as node 1 in this paper. Note that the insertion of node 1 as the external node is done with the condition that there is at least one biconnected component of the circuit graph that has more than two nodes connected to the exterior. If this condition is not met, then there is no need to introduce the extra external node. For the simplicity of the presentation, the remainder of this paper considers that the condition holds, as the opposite case is more simple and is treated similarly.

The Algorithm

The algorithm has nine phases that we present in the following paragraphs.

Phase 1: Forcing a Planar Embedding

In this phase, the search for a planar embedding of the input graph is carried out by a recursive procedure derived from [3]. This procedure planarises the graph by deleting a minimal set of edges if needed (see Figure 3). As the set of edges is not unique, two types of control upon edge selection are used. The first is done via the initial ordering of the adjacency lists that represent the graph (first path in Figure 3). The second type of control is done via the decision of which edge to delete in a crossing pair. An edge weighting function is used to select the appropriate one. Each time an edge is deleted, we restart the procedure on the biconnected components of the new graph. This procedure ends when the graph is planarised, producing a list of planar biconnected components, a list of deleted edges and a list of the generated paths as if they were produced by **pathfinder** [11].

Phase 2: Face Finding

In this phase, the faces of a planar embedding are generated. As several embeddings may exist, the current procedure is exhaustive in its search. This permits backtracking to generate different RACG solutions. The paths generated in the planarisation phase for each biconnected component are checked for conflicting embeddings. The checking is based on the path interaction lemmas presented in [11]. This permits the assignment of "same" or "different" embedding for the pairs of paths. These assignments are input to a procedure which generates the faces. This procedure outputs the external face and the set of internal faces that define the planar embedding for each biconnected component. For example, paths (5,3) and (4,2) in Figure 3 should always be on the opposite side of the cycle (1,2,3,4,5,1). The positioning of one of these two paths on the left or right side of the cycle above will lead to a different embedding and different generated faces.

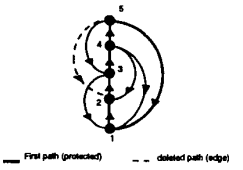

Figure 3: Planar embedding of graphs. The first path (1,2,3,4,5,1) is protected. For the embedding in the figure, the removal of edge 5-2 will planarise the graph. An alternative solution is the removal of edge 3-1.

Phase 3: Merging the Biconnected Components

In this phase, the planar biconnected components are merged together. The algorithm carries out this step as follows. A biconnected component host is selected. If node 1 is the external node then the biconnected component containing it is selected. Then for each of the remaining components we find all the faces in the host that contain the articulation node (node 8 in Figure 4). We select as a host the face that maximises the number of deleted edges between its nodes and the biconnected guest, where we insert it. If several faces satisfy this selection criteria, then we look for the face (in the path) that minimises the cost of restoring the most expensive deleted edge (see Phase 4 for more details on edge restoration). The external face of the guest is combined with the hosting face to produce a set of new faces. The hosting face is replaced by the set. In addition, a defacto edge is inserted to make the graph resulting from the merging a single block.

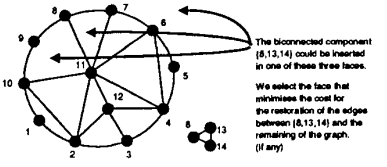

Figure 4: Biconnected components merging.

Phase 4: Restoration of Deleted Edges

This phase restores deleted edges using one of three options. In the first, the algorithm gives the user the possibility of restoring the deleted edges using techniques other than the insertion of wiring blocks. In the second option, the deleted edges are restored by the insertion of wiring blocks each time a crossover occurs. In the third option, adjacent crossovers are grouped into one wiring block. The deleted edges restoration algorithm uses a "branch and bound" search to determine an appropriate restoration path. For each edge to be restored, the restoration algorithm finds the "faces path" that separates the endpoints of the edge (see Figure 5). The path selection is based upon the least edge weight separating two adjacent faces. Then, the edge is restored with the injection of wiring blocks at its crossing with the edges in the faces path. The faces of the embedding are updated by this operation.

Phase 5: External Defacto Connections

The faces generated in the previous phase and containing node 1 as external node are checked for their degree. If such a face has a degree

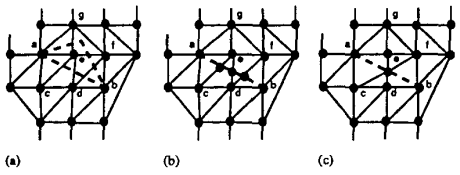

Figure 5: Left: Two different faces paths to restore the edge (a-b): (a,e,c), (e,c,d), (e,d,f), (f,d,b) and (a,g,e), (g,e,f), (e,d,f), (f,d,b). Right: The first faces path is selected and wiring blocks inserted on edges crossings.

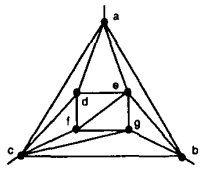

Figure 6: An illegal cycle of length 3: (a-b-c). To destroy this cycle is the same as restoring a zero weight edge between one of (d, e, f, g) and the external face and injecting a wiring block at the crossing with one of the illegal cycle edges.

greater than 3 and a set of its nodes not connected to the external node, then the nodes in this set will have a forced (defacto) connection to the exterior. This face will be destroyed (generating new faces) by the introduction of new edges between these nodes and node 1.

Phase 6: Node Shortfall Solving

In this phase we solve the shortfall of node degree (node degree < 3). The algorithm proceeds as follows. First, it tries to increment the degree of a node by injecting a defacto edge if the node belongs to a face with degree > 3. The node with minimal degree in the face is selected for the defacto edge. If such a face does not exist, then a neighbouring face of a face that the node belongs to is found using least cost branch and bound search. Then the node with least connection in that face is selected as the end node of an edge that the algorithm installs using wiring blocks.

Phase 7: Forcing a BNG Path

The biconnected components generated in phase 1 have the property that node 1, the exterior node, could not be an articulation node for any pair of biconnected components. This is the result of the DFS operation and the condition for the insertion of node 1 as an external node as described above. This result is used in the following manner. First consider B_e to be the biconnected component including the external node, and let P_b be the BNG of the node set ($B_e - external\ node$). If P_b is a path then no further action is taken. If P_b is not a path then we search in B_e for all faces that have a degree ≥ 4 and we destroy them by creating a defacto adjacency. The destruction of the faces will ensure that P_b becomes a path. According to the BNG, a list of valid user corners is built for later use. The criteria of validity at this stage concerns only the structure of the BNG. Each user corner that is an articulation node, or that is not in the extremities blocks of a BNG path (path length $>$ 1) is considered invalid.

Phase 8: Illegal Face and Cycle Destruction

As stated in the RACG conditions earlier, faces not adjacent to the periphery should have a degree of 3 and thus faces with degree ≥ 4 have to be destroyed. To do this, we find in the faces that do not contain node 1 (the exterior node) the nodes that have a shortfall and we establish defacto edges between them. This is applied until no more faces are illegal. In addition, cycles that have a length ≤ 3 and that are not faces have also to be destroyed. An example of such a cycle would be enclosing faces. To destroy these cycles, the algorithm uses first a DFS algorithm to find them. Then a wiring block is inserted in the weakest edge of each cycle (see Figure 6). Two additional connections are added to the wiring block, one on each of the sides defined by the cycle. If the edge is on the periphery, then one of these two connections would be with the exterior. The illegal cycle removal is performed after the illegal faces destruction. This is to avoid the creation of unwanted cycles by the insertion of defacto edges when destroying faces.

Phase 9: Selecting the Corners

In this phase, we find first the shortcuts and the CIPs (or CCIPs, depending on the BNG). The algorithm then checks if any of the valid user corners belong to the CIPs, and the corresponding CIPs are removed. If the user selected a node to occupy two corners, then only one is searched for in a CIP. If the graph's BNG is a path of length > 1, then only the corner nodes at the path extremities are considered, and as stated previously, the other corners would have been revoked in the BNG path forcing phase. After that all CIPs (or CCIPs) are satisfied, if any shortfall of corners still exists then valid double corners are considered before the consideration of nodes from the external faces to adjust the number of corners to 4 (or 2 for each extremity when dealing with CCIPs). At the end of this phase, the graph will be an RACG and ready for dualisation.

ALGORITHM EVALUATION

As mentioned previously, the algorithm was implemented as part of a library of algorithms used by PIAF, a knowledge-based/algorithmic floorplanning system. It is written in Pascal and runs on a Vaxstation 2000 running VMS. The RACG produced by the algorithm is dualised by an algorithm based on [9] and developed by the author. The input to the algorithm (the edge weights) can be used to influence the solutions according to design needs. Experiments with the algorithm on real graphs (FBDs) have shown that the number of wiring blocks injected for communication restoration is slightly larger than in the case where wiring blocks are only injected for node shortfall solving and/or illegal cycle destruction. Another approach to block insertion would consider the use of previously inserted blocks in the routing. The algorithm could easily be modified to do so. The performance of the algorithm is satisfactory as shown in the remainder of this section. The wiring blocks inserted by the algorithm are named "paf_wb_N", where N is a number.

Example 1: A Simple Graph

The graph of this example is shown in Figure 7. It is planar and contains 16 nodes (exterior excluded). The corresponding rectangular dual of the algorithm's RACG is shown in Figure 8. As can be seen, no wiring blocks are inserted because of the planarity of the input graph and the fact that all nodes have an acceptable degree.

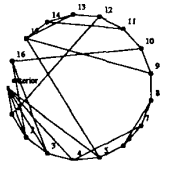

Figure 7: Plantest: A planar graph with 16 nodes.

Example 2: The Encryption Chip

The graph of this example is shown in Figure 9 and represents an encryption circuit. The RACG and a rectangular dual, with the third option

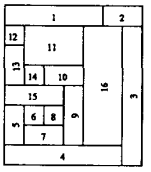

Figure 8: A rectangular dual of the RACG generated for the planar graph shown in Figure 7.

enabled to planarise the graph are shown in Figures 10 and 11 respectively . Note the presence of wiring blocks as a result of planarisation and node degree shortfall.

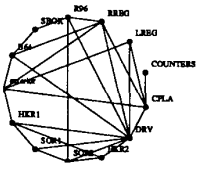

Figure 9: The encryption chip input graph.

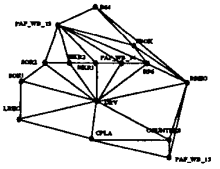

Figure 10: RACG of the graph shown in Figure 9. It was generated using the third option (wiring blocks grouped). Blocks on the external cycle connect to the I/O pads.

Run Time Examples

Table 1 shows the run time in milliseconds for the two examples presented above and two other non planar graphs, K_5 and $K_{3,3}$. When edges restoration is disabled, the algorithm has shown an expected peak run time proportional to N*M, where N and M are the number of nodes and edges of the input graph respectively. When enabled, local optimisation ("Branch and Bound") processing for edges restoration makes the definition of a timing model difficult.

CONCLUSION

An algorithm for building graphs admissible for rectangular dualisation has been presented. The algorithm offers three options for communication crossover solving, wiring blocks, grouped wiring blocks or passthrough. The grouping option in particular prevents fast increases in the number of wiring blocks in the circuit. The algorithm is efficient and is very well suited for interactive applications.

ACKNOWLEDGMENTS

The author is grateful to David Skellern, Peter Nickolls and David Myers for their encouragement and acknowledges the support of an Australian Commonwealth Postgraduate Research Award.

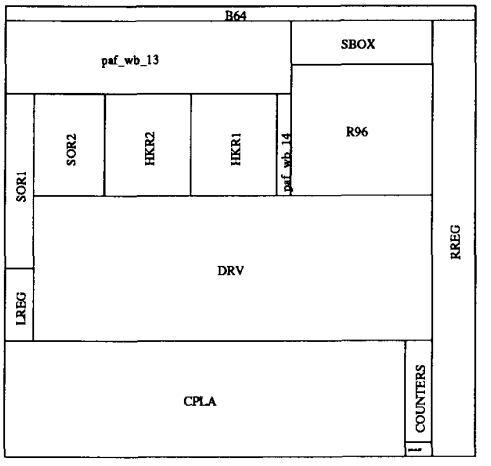

Figure 11: A rectangular dual of the RACG shown in Figure 10.

Table 1: Approximate CPU time for some graph examples (in ms).

Graph	Planar Embedding	RACG Building (Total)
Example 1	600	810
Encryption chip	890	1880
K_5	800	1180
$K_{3,3}$	610	780

REFERENCES

[1] Jayaram Bhasker and Sartaj Sahni. A linear algorithm to find a rectangular dual of a planar triangulated graph. In *Proc. 23rd Design Automation Conf.*, pages 108–114, June 1986.

[2] W. Heller, G. Sorkin, and K. Maling. The planar package planner for system designers. In *Proc. 19th Design Automation Conf.*, pages 253–259, 1982.

[3] J. Hopcroft and R. Tarjan. Efficient planarity testing. *ACM*, 21(4):549–568, October 1974.

[4] M.A. Jabri and D. Skellern. Inference of efficient integrated circuit floorplans using PIAF. Accepted for publication in IEEE Expert.

[5] M.A. Jabri and D.J. Skellern. High level knowledge-based communication solving for VLSI circuits. In *Proc. Microelectronics Conf. VLSI 87*, pages 50–55, Melbourne, Australia, April 1987.

[6] M.A. Jabri and D.J. Skellern. Implementation of a knowledge base for interpreting and driving integrated circuit floorplanning algorithms. *International Journal for Artificial Intelligence in Engineering*, 2(2):82–92, April 1987.

[7] M.A. Jabri and D.J. Skellern. A mixed knowledge-based/algorithmic approach to custom integrated circuit floorplanning. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 289–292, 1986.

[8] K. Kozminski and E. Kinnen. An algorithm for finding a rectangular dual of a planar graph for use in area planning for VLSI integrated circuits. In *Proc. 21th Design Automation Conf.*, pages 655–656, June 1984.

[9] K. Maling, S. Mueller, and W. Heller. On finding the most optimal rectangular package plans. In *Proc. 19th Design Automation Conf.*, pages 663–670, 1982.

[10] R. Otten. Automatic floorplan design. In *Proc. 19th Design Automation Conf.*, pages 261–267, 1982.

[11] Robert E. Tarjan. *An Efficient Planarity Algorithm.* Stan-CS-244-71, Computer Science Department, School of Humanities and Sciences, Stanford University, 1971.