The CSI Journal on Computer Science & Engineering

The CSI Journal on
Computer Science and Engineering
Vol. 1, No. 2(b), Summer 2003
Pages 27-38
Regular Paper

A Specification-Driven Framework for the Floorplanning and Placement of Hierarchical VLSI Designs

Morteza Saheb Zamani

Computer Engineering Department
Amirkabir University of Technology, Tehran, Iran

Abstract

With the growing complexity of VLSI systems, automatic physical design of today's systems has become very complex. One approach to control overall complexity is to divide the design into several levels of hierarchy. This may produce poor results if the inter-dependence between various components of the design is not considered. In addition, the high degree of inter-dependence between design processes operating at the physical and higher levels of abstraction (e.g. the specification level) necessitates a number of iterations between these levels. One effective solution, exploited by this work, is the use of a common design hierarchy for both specification and physical level design. This can provide physical data to the specification level synthesis process, thereby reducing the design cycle. In this paper, a framework for the floorplanning and placement of macrocell designs is introduced. Inter-dependence between levels of a design specification hierarchy is taken into account in this framework. Since the framework is not inherently restricted by either the hierarchy depth or the hierarchy branching factor, it is able to preserve an arbitrary specification hierarchy. The framework gathers geometric information about the design over several traversals of the design specification hierarchy and sets the physical geometries, such as port positions, orientation, etc., in a stepwise refinement fashion. The results show that the framework produces good results quickly for large designs.

Keywords: VLSI, physical design, floorplanning, placement, specification hierarchy

1. Introduction

The process of VLSI design is often performed at different levels of abstraction. At the *behavioral* level, an algorithmic view of the design is specified and a high-level synthesis process produces the schematic of the design at the *structural* level or the *register transfer level (RTL)*. The structure of the design is typically described in terms of its functional components (modules) and their interconnections, the so called *netlist*. A *physical design* process then determines the physical (geometrical) characteristics of the design components, such as the module dimensions, the interconnection paths and the placement of modules on a two-dimensional plane. The physical design is used to generate the fabrication masks for the manufacturing processes.

Since the physical design tasks are complex, they are usually decomposed into several simpler subtasks, including partitioning, floorplanning, placement, routing and compaction. This paper focuses on the floorplanning and placement processes which determine the sizes, shapes, positions and orientations of all modules in the design given the structural specification of the design. The criteria to be considered by these processes typically include minimizing connection lengths and the design area, as well as meeting some additional design constraints such as aspect ratio and performance.

Due to the combinatorial nature of the processes which solve these problems, finding an exact solution is not generally practicable within a reasonable time, so heuristic approaches are usually proposed which produce acceptable results. One

of the powerful methods used to control the overall complexity of the processing is the decomposing of the designs into subproblems with a small number of modules.

This is often done at the higher levels of abstraction resulting in a hierarchical design specification. In such a specification, the design may be partitioned into many levels containing nodes, where each node represents a module which is either a leaf module or a parent module containing further nodes. Modules may interact with each other through input/output connections and ports. In the physical design, however, high levels of interaction between modules can result in high routing cost and degraded performance if simple *divide and conquer* heuristics are used without considering these costs. In this paper, hierarchical approaches to floorplanning and placement are proposed, which consider the inter-dependence between different parts of a hierarchically specified design.

On the other hand, there is also process inter-dependence between the specification and realization of the design which usually leads to many iterations between these processes over the various levels of abstraction. For example, the high-level synthesis process (whether manual or automatic) may produce a design structure which is considered to be poor only after the physical design is performed. Therefore, the structural specification may need to be modified each time the physical design fails to meet some constraint, such as performance, area, aspect ratio, and/or power consumed. The cost of iterating between these processes can be very high in practice if the layout result cannot provide helpful information to the higher level synthesis process. One way to provide bilateral information between these various design processes is to use a common specification hierarchy[1]. Therefore, an objective in this work has been to preserve the specification hierarchy.

There are several benefits in maintaining the specification hierarchy, specially for large system designs. Reducing the number of iterations between processes operating at the different levels of abstraction, facilitating the modification of the design, and easier debugging of the design at physical level, are among the reasons for the interest in maintaining the specification hierarchy. Notwithstanding, if the specification hierarchy is not suitable for physical design, a flattening/repartitioning pre-process may produce an appropriate hierarchy for the framework presented here.

2. Prior Work

There has been a large amount of research in the field of *single-level* floorplanning over the past decades [1,2,3,4,5,6]. However, multi-level hierarchical floorplanning of macrocell designs and the inter-dependence among the hierarchy levels have not been studied closely. One of the main reasons why this problem has not been addressed is that macrocell-based designs have not had many modules until recently and the difficulty in handling inter-level dependencies might have hindered CAD designers to adopt this strategy.

However, in the recent years, due to several reasons, including growing complexity of designs and the need for design reuse, the number of macrocells in a design is becoming very large. Therefore, physical design tools have to handle very complex designs with many hard macros [7].

However, there are few approaches which attempt to perform floorplanning hierarchically. [8,9,10] enumerate all possible topologies of every cell at each hierarchy level and then select those that best satisfy the objective function for placement and routing within a module incorporating the cells. Dai and Kuh's algorithm [10] tries all possible templates in a top-down process to find the best topology according to a desired chip aspect ratio and I/O pad positions. However, since this approach fails to take module shapes into account at the higher hierarchy levels, it does not work well for chips containing cells with fixed geometries and it may need considerable backtracking to effectively search the solution space. Eschermann et al [8] propose an improvement on this approach which passes shape estimation information up the hierarchy levels from the bottom. This information is then utilized by a process operating top-down, to evaluate the various topological possibilities (TPs). This bottom-up process largely constrains the search space since it only computes the shape of one configuration at each level, by considering such measures as area and shape, and does not carry enough information about the shapes of the modules from the bottom levels. Pedram and Preas [9] propagate accurate information by enumerating all TPs in a bottom-up traversal and labeling each module with its shape function (the function which gives the possible dimensions a module can have). Some other techniques, such as [11] and [12] also use shape functions to pass size information to upper levels of the hierarchy.

These approaches are limited by the number of modules at each level, as well as the number of hierarchy levels, able to be considered, due to the fact that the number of TPs grows exponentially with the number of modules. For example, a five module cell may have more than 300,000 different TPs [8].

This paper presents a fast algorithm which is capable of floorplanning followed by placement and routing of very large hierarchical designs. It considers the effects of the hierarchy levels on each other and takes routing into account during the floorplanning process.

3. Problem Definition and Solution Modeling

The system introduced in this paper forms a framework for floorplanning and placement into which routing can be integrated. The framework is able to handle large designs with an arbitrary hierarchy. In the adopted model, the input to the system consists of:

- a design hierarchy,
- the netlists of all parent modules in the design,
- the position of I/O ports at the top most level of the hierarchy (optional),
- the size and shape of the leaf level modules, and
- the port positions of the leaf level modules.

The algorithms for floorplanning and placement are to determine the following characteristics of a design, so that a

combination of the area of the design and the estimated connection length is minimized:

- the position of all modules in the design,
- the shape of all parent modules,
- the port positions of all parent modules except for the top level module whose port positions can be specified as input to the algorithm, and
- the orientation of fixed shape modules (leaf modules).

The difference between placement and floorplanning is that the modules in the floorplanning problem are treated as having flexible shapes and port positions (*i.e.* the parent modules in our model) and the floorplanner should determine these characteristics for each module, whereas the modules which the placer works with have fixed shape, size, and port positions (leaf level modules).

The two common criteria, namely total area and total connection length, are used as the objects of optimization. However, other criteria like design performance can be considered in the algorithms, with minor modification.

Another objective of the floorplanning and placement algorithms in the present model is to preserve the specification hierarchy in the final layout, unless there are reasons for changing it. In the latter case, the specification hierarchy is converted to the desired specification hierarchy and the system attempts to realize the physical design whilst maintaining the new hierarchy.

4. Floorplanning Hierarchical Designs: Motivation and Algorithm Overview

The placement of modules at a given level of hierarchy can be performed if the size, shape and port positions of the modules at that level are known. In a hierarchical environment, initially these characteristics are generally unknown and depend largely on the geometrical characteristics of the parent, sibling and children modules. The characteristics of an arbitrary module should normally be bound by the floorplanning, placement and routing processes of modules at lower, same and higher levels of the specification hierarchy, except for the top or bottom most levels where such information is usually provided as part of the module specification or constraints.

For instance, an estimate of the interconnection wire lengths associated with a module at some intermediate level of the specification hierarchy is essential for many algorithms to be able to find the optimal placement of the submodules of this module. However, this cannot be done until the optimal positions of ports on the module boundary are determined by a port assignment process, following the placement of the upper level module to which this module belongs (*i.e.* its parent module). On the other hand, the placement of the parent module may not be determined optimally until the size, shape and port positions of its submodules, including the module under discussion, are known. Unfortunately, this information (size, shape and port positions of the submodules) can only be known accurately *after* the placement and routing of the submodules are finished.

The port positions in our algorithm are known for the module at the top most level of the hierarchy (root module), although this knowledge is not necessary. Furthermore, the size, shape and port positions of the bottom most level modules are explicitly specified as input to the algorithm. This information is used to direct the floorplanning process.

Based on the above discussion, a floorplanning process may be directed by considering either the leaf cells (bottom-up) or the specification (top-down) first. The bottom-up approach has the advantage of using accurate information (size and port positions) about the modules to be placed. However, since this approach ignores connectivity information of modules in the upper hierarchy levels, an optimal placement (packing) found at one level of the hierarchy may not be optimal with respect to the higher levels, and therefore, the whole design. For example, in the circuit shown in Figure 1, even if the placements within module 1 and module 2 are optimal, they may result in long connections and poor space utilization at the upper levels of the design.

On the other hand, if the algorithm starts from the specification, even though a good global interconnection view is available, there is no information about the module sizes since the placement and routing internal to the modules has yet to be determined.

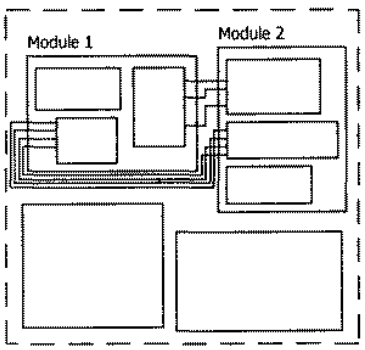

Figure 1. Inter-dependency between levels of hierarchy

Due to such inter-dependence between different parts of the problem, a simple *divide and conquer* heuristic *per se* cannot be applied and a large amount of information occurring across adjacent hierarchy levels needs to be considered before the final placement and routing are determined. In the framework presented in this paper, during the information passing processes, floorplanning is done in a stepwise refinement fashion with the information required at each stage being extracted and passed to processes operating at the next level of the specification hierarchy. The information passed is a set of constraints which is generated, at the previous steps, in such a way as to avoid over-commitment too early when sufficient information has yet to be gathered. Premature decision may over-constrain the tasks at the next hierarchy level and produce poor results (that is, the design is forced into a poor local minimum within the design space).

This framework begins from the top level of the hierarchy and initially sets the shapes of modules within each level to be the same. This is done because no shape information is available at the initial stages. The sizes of all parent modules

are very roughly estimated, in a preprocessing bottom-up traversal of the specification hierarchy, by calculating the composed sizes from all leaf modules within each underlying branch of the hierarchy tree. The algorithm then performs an initial placement of modules at the top level, guided by the I/O port placements of the parent module at this level. This is followed by the assignment of I/O ports on each placed module's boundary, in an attempt to optimally route all interconnections at the parent module level. The process now recurs for each of the placed modules.

This top-down process produces an initial floorplan containing the global connection paths for all modules in the design, which are locally optimum. Based on this information, a bottom-up process is then performed which produces a better estimation of the module sizes, successively, at every level up the specification hierarchy. Each module size is calculated based on the shapes and placement of its constituent modules. These placements are performed in this bottom-up process, whilst taking into account the information obtained (that is the interconnection harness) from the preceding floorplanning step.

This is followed by another top-down process which gives a more accurate global view of connections, since it uses the more realistic module size information calculated in the previous bottom-up pass. Like the first top-down pass, at each level, the port positions are set for each submodule which then act as constraints on the modules at the next level as this second top-down pass progresses.

The final bottom-up pass then uses all the information computed in the previous passes to perform the actual placement and routing. In the final pass, the placement of the modules at each level is attempted by considering the parent module's port positions, which carry the information about the global connectivity. In this way, a blind bottom-up process which results in long wires as in Figure 1 is avoided.

Finally, a reshaping process may be applied to reduce the wasted space attributable to module shape mismatches at any level in which the wasted space is considered to be significant.

5. Algorithm Details

5.1 Initial size estimation: the preprocessing bottom-up pass

In this pass, an approximate size estimate of all modules in the design is calculated. The processing starts from the bottom most level where the exact size information is available, since leaf cells have fixed sizes. The size of a module at a given hierarchy level is estimated to be the sum of its child modules' sizes. This estimate is inaccurate as the size of a module depends heavily upon the configuration of its submodules. Nevertheless, since there is no such information initially, this rough estimate of size is used at the beginning to formulate an initial floorplan. The size estimation procedure is applied recursively to all modules during the bottom-up traversal of the hierarchy tree.

5.2 Initial floorplanning: the first top-down pass

The goal of this step is to obtain interconnection paths which are ideal within each local hierarchy level. To do

this, it is necessary to determine the relative positions of modules within each parent module. The module sizes are taken from the estimations made during the preprocessing bottom-up pass.

In this stage, modules are considered as circles, since there is no information yet available concerning module shape. Since the sizes (area) of the circles are proportional to the estimated sizes of the modules that they represent, they determine the sizes of the submodules, to be placed at each level of the specification hierarchy, relative to each other, and bounded by the parent module size. Based on the modules' sizes and logical I/O port configurations, it is possible to determine the optimal position of each submodule's ports from connections with its sibling submodules and parent module.

Representing a module as a circle also avoids the need to fix I/O ports to some side of a rectangle at an early stage of floorplanning. The simplicity of the algorithm which approximately places modules guided by locally optimum wiring is another reason for representing modules as circles, in this stage.

Using the circle representation, the task of finding the relative positions of modules at each level can be viewed as an optimization problem with an objective function which maximizes the size of submodules within a parent module circle, so that each module maintains its relative size ratio and port configuration and the total connection length within each hierarchy level is minimized. Figure 2 illustrates the result of this pass at one hierarchy level. The lines connecting points represent either a (set of) wire(s) between the ports of two modules or between a module's ports and its parent module's ports.

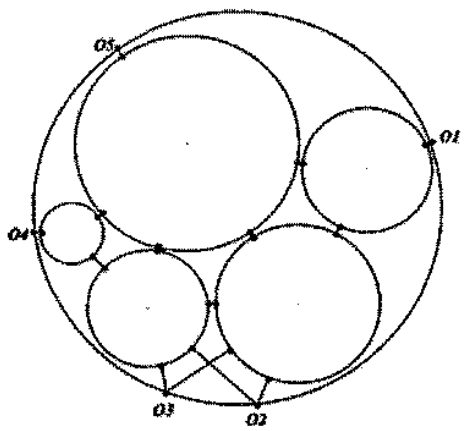

Figure 2. The placement of circles at one level

The processing starts at the root module of the specified hierarchy, where port positions on the boundary correspond to desired I/O port positions. After placing all submodules and calculating their maximum allowed sizes relative to the parent module, optimal port positions for each module of this

parent are determined by connecting the centers of the corresponding circles (see Figure 2). This procedure is applied recursively to each set of modules at each hierarchy level, until the leaf modules are reached. Routing paths can now be determined at each hierarchy level, as port position information is passed from the top to the lower levels of the design. Figure 3 shows the result of the process for a 3-level hierarchy.

Note that in this pass, the module sizes are merely the *relative* sizes with respect to each other and the parent module. The relative sizes at each level are sufficient to determine the relative placement of the submodules and thence, their port positions which are locally optimum with respect to the current parent module. Therefore, starting from the top level, there is no need to set an initial size for the entire design. The size of the root module is simply set to one and all module sizes are considered relative to it.

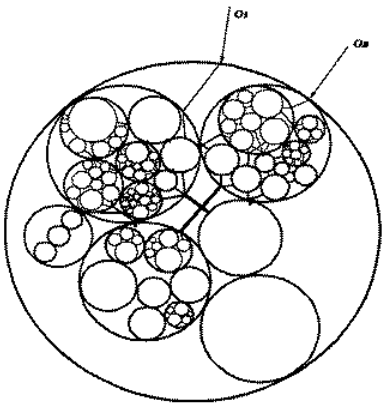

Figure 3. The connection paths produced by the initial floorplanning

5.3 Improved size estimation and propagation: the second bottom-up pass

At the leaf level of the hierarchy, accurate information about module sizes is available. This bottom-up tree traversal attempts to propagate more accurate size information to the upper levels of the tree. The size information is based upon the relative placement of modules determined by the first top-down pass and therefore, more accurate than the estimation calculated in the preprocessing bottom-up pass.

Each module is modeled by a circle having an area which is the minimum space required to place the module without having to fix its orientation. To be exact, for 90° rotations, the minimum required space to place the module in either orientation is a cross shape consisting of the module itself and its 90° rotated version (see Figure 4). Using a composite rectangular and rotated rectangular shape makes the algorithms complicated, hence the circle representation was used. The price to be paid for this simplification will be inaccuracy in size estimation for the modules with large aspect ratios.

Routing area is estimated and incorporated into the size of each module's circle. The routing area is proportional to the number of ports a module has. It is necessary to consider the routing area around each module at the early stages of floorplanning since adding this space at the end may result in a large amount of wasted space in the design. As may be seen in Figure 5, very compact designs without the consideration of routing area may result in large dead spaces after adding routing area.

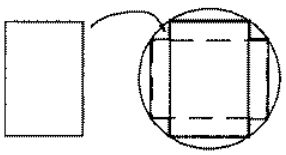

Figure 4. The circular boundary is the minimum space required to represent a rectangular module with an unknown orientation. The cross shape (highlighted rectilinear region within the circle) is the minimum space required to represent a rectangular module with an orientation restricted to 0°, 90°, 180° or 270°

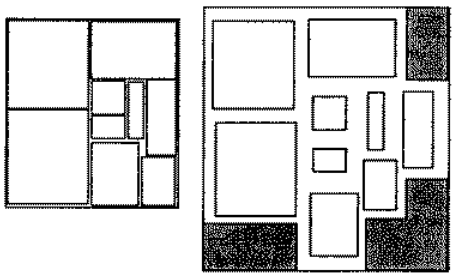

Figure 5. A compact placement without the consideration of routing area (a) may result in large dead spaces after adding routing area (b)

Starting from the leaf modules, the size of each parent module is estimated based on the relative positions of its submodules. These are calculated from the circle (module) placements, having fixed the external port positions, which are dictated by the interconnection paths obtained during the first top-down pass. The port positions are described in terms of the angles at which the wires connect to the module, and are, therefore, independent of the module's size. These are used, in turn, to guide an optimal placement of submodules within each parent module.

The placement procedure in this traversal is similar to the one used in the floorplanning pass, but with fixed submodule sizes and variable parent module sizes. (In the previous top-down pass, the parent modules had fixed sizes and the submodules sizes were variable by a constant factor, - see Section **Error! Reference source not found.**). The size of each parent module is now determined as the minimum circle enclosing all its submodule circles (see Figure 6). By applying this procedure recursively in traversing back up the specification hierarchy, the sizes of all parent modules in the design are estimated.

5.4 Floorplanning improvement: the second top-down pass

The information obtained during the initial floorplanning pass is inaccurate due to the simplistic module size estimation. With better estimated sizes available after the second bottom-up pass, a procedure similar to the first top-down pass is carried out with the more realistic module sizes to obtain more accurate interconnection path estimates.

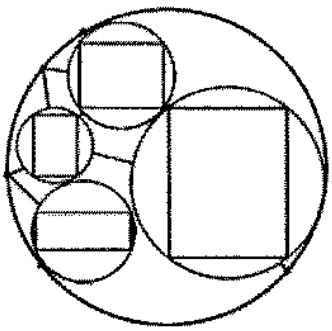

Figure 6. The size information propagation

5.5 Actual placement and routing: the third bottom-up pass

At this stage, where information about the wiring paths, port positions, and module sizes is available, the final placement is performed using rectangular blocks. The algorithm for determining the relative placement of circles is based on the last calculated port positions. In this way, the upper level port positions, which carry the information about the environment in which each module exists, affect the placement of the submodules at each hierarchy level.

The rectangular blocks (modules) are then mapped into their corresponding circles. Until now, the orientation determination had been postponed in order to gather more information. The mapping involves determining the best orientation (x and y mirroring, and 90°, 180° and 270° rotations) of the blocks by enumerating the eight possible ways of positioning each block inside its corresponding circle and choosing the orientation which results in minimum length routing when connecting the modules' ports to the port positions around their corresponding circles (see Figure 7). If the port position information was not available, orientation determination would be an NP-complete problem, since for N modules, 8^N possible orientations should be tried to determine the best orientation of the modules for their environment (interconnection with other modules). However, by applying the constraints which characterize the environment, the block orientations can be determined in linear time with respect to the number of modules at each level.

Although it has been assumed that leaf modules have fixed geometries, the algorithm is able to select from various versions of these modules providing their geometries do not overlap with other modules after their orientation mappings.
As there is space left after the circle to rectangular block conversion, a two-dimensional compaction algorithm is applied. However, dead spaces will inevitably remain after the compaction process due to the fact that the compacted shapes may not be rectangular when propagated to the upper levels of the hierarchy. This problem may be ameliorated by adjusting module shapes to make them closer to rectangular by trading off wiring length for wasted area. The method used at present is based on simulated annealing placement[13,14], with a restricted move set and low starting temperature which preserves the relative placement and results in fast compaction.

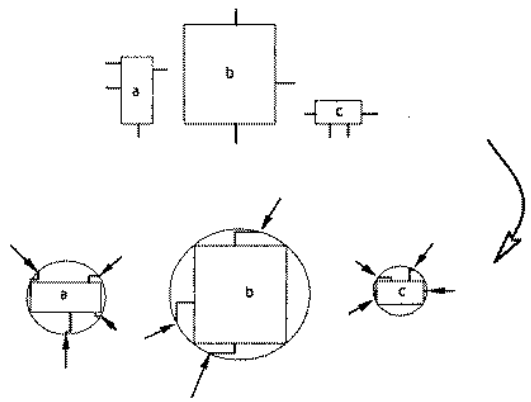

Figure 7. Setting the block orientations; from the eight possible orientations (90° rotations and x and y mirrorings in the horizontal and vertical directions), the algorithm selects the one which best matches the environment in terms of wiring length

The routing of the oriented modules can now be performed. The internal connections can be routed using some global and channel routing algorithms. The connections between the external ports of modules are better postponed until all the modules to which they are connected are known. In this way, during the placement and routing of the upper hierarchy levels the ports may be assumed to be located somewhere inside the modules, actually on the boundary of a submodule. This gives a great deal of flexibility and avoids wires being unnecessarily routed around some modules. In this process, port assignment, which is inherently top-down, is performed in a top-down fashion from the level at which all port locations of a net are known, to the lowest level. Figure 8 shows an example of port assignment in a middle level of a design hierarchy. The two ports on module M1.1.1.1 and module M1.3.1.1 which are to be connected, are left unrouted until module M1 is fixed by determining the locations of modules M1.1, M1.2 and M1.3. Now the inter-submodule connected port positions are calculated optimally for the submodules of M1.1 and M1.3, down to the level where those ports exist. Note that it is not necessary to postpone this process until further port resolutions have been made at the upper levels since the routing of such connections can be performed as soon as the placement of the modules contributing to these connections is determined.

It is worth mentioning that the ports of the upper hierarchy level modules are distinct from the ports of the lower levels in the abstract specification. Therefore, when the assignment of a port is postponed, the corresponding port at the upper level of the hierarchy is assigned the same position as the lower level module's port until the optimal position on the

boundary of the module at the corresponding hierarchy level is determined (see Figure 9).

When the number of external connections in a module is large relative to the internal connections, the above port assignment strategy may lead to some problems. For example, in this case, routing area estimation may be difficult since the connection paths to external ports are not known during placement. A probabilistic approach, such as that proposed in [9] may be used to ameliorate this problem.

Moreover, postponing the routing of external connections, when there are large numbers of them, may produce sub-optimal routing results since routing all the internal and external connections separately may over-constrain the routing process. Notwithstanding, it is normally expected that the specification hierarchy is constructed (either by the designer or by a repartitioning process) in such a way that the heavily connected modules are not in different parent modules, and hence the number of external connections are normally low relative to the internal ones. The above issues are interesting problems for further research. When the placement and routing of a module is done, the minimum bounding rectangle of each compacted rectangular module is determined for the parent hierarchy level, and the procedure is recursively applied upward until the final layout of all modules is determined.

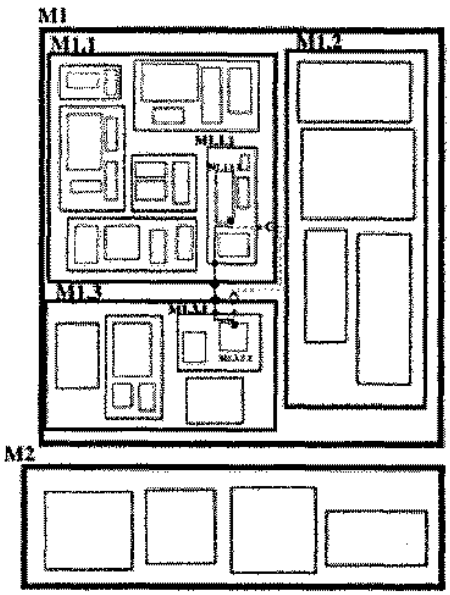

Figure 8. Deferring assignment of some ports which have connections to modules at other hierarchy levels to determine the best path at later stages. The dashed line shows an alternative path if a premature decision on port assignment (empty circles) is made

5.6 The placement algorithm within one module in the specification hierarchy

The placement algorithms for all passes should have sufficient commonality to enable information obtained from one pass to be used in the following passes. For example, if the placement procedure used in the first top-down pass is based on the min-cut algorithm but the one in the second bottom-up pass uses a force-directed algorithm, the configurations resulting from one algorithm may be quite different from the other and, hence, the port position information which is derived from the former may misguide the latter processing passes.

The placement algorithm should also be fast and not require a huge effort to generate an initial configuration of the design since the information used in the early stages of the whole process is inaccurate. The placement algorithm should also be able to take into account external connections with fixed positions. Some algorithms do this by considering the external ports as zero size modules, set at fixed positions.

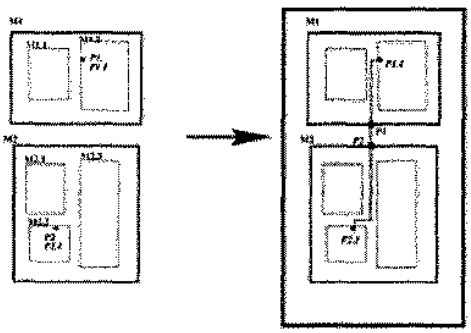

Figure 9. The assignment of ports on the boundary. Ports *P1* and *P2* belong to the modules M1 and M2 respectively and ports *P1.1* and *P2.1* belong to the modules M1.2 and M2.2 respectively in the specification

The heuristic used here to perform the placement of circles is based on the force-directed method [15]. This algorithm first determines the positions of the modules' centers by calculating the equilibrium point when each connection is considered to be a spring-like attractive force. Note that, at all levels of the hierarchy, after the top-down passes, every module has fixed external port positions on its circular boundary. When the port positions are not specified for the top (root) level, a set of repulsive forces between modules with low connectivities can be introduced to avoid the modules being superimposed at a single point. The port positions are then set optimally on the boundaries, after placement, so that the interconnection lengths to the ports are minimized. The problem can be formulated as solving the following equations in order to determine the optimal positions of the module coordinates (x_i, y_i):

$$x_i = \frac{\sum_{j=1}^{E} C_{o_j,i} \cdot X_{o_j} + \sum_{j=1, j\neq i}^{N} C_{i,j} \cdot x_j}{\sum_{j=1}^{N} C_{i,j} + \sum_{j=1}^{E} C_{o_j,i}} \quad (1)$$

$$y_i = \frac{\sum_{j=1}^{E} C_{o_j,i} \cdot Y_{o_j} + \sum_{j=1, j\neq i}^{N} C_{i,j} \cdot y_j}{\sum_{j=1}^{N} C_{i,j} + \sum_{j=1}^{E} C_{o_j,i}} \quad (2)$$

for $i = 1, ..., N$, where

N = number of submodules,
E = number of external ports,
(X_{Oj}, Y_{Oj}) = external port j coordinates,
$= (R\cos\theta_j, R\sin\theta_j)$
R = parent circle radius,
θ_j = radian angle of external port j on the parent circle,
$C_{Oj,i}$ = connectivity between external port j and module i,
$C_{i,j}$ = connectivity between module i and module j,
(x_i, y_i) = module i coordinates.

When the relative positions of circle center points are determined[2], the maximum size of each circle without overlaps is calculated by a *closest pair* algorithm [16]. This is straightforward for circles of equal size. Given the coordinates of a set of points in a two-dimensional plane, the algorithm looks for the point pair which has the least Euclidean distance. The simplest algorithm is to check all the point pair distances and identify the smallest one. The time complexity of this algorithm is $O(N^2)$ but the algorithm presented in [16] has time complexity of $O(N\log N)$.

For unequally sized circles, the closest pair algorithm can be used, providing the definition for the distances between point pairs is scaled by the relative sizes of the corresponding circle pairs. The relative sizes of the circles at each node l in the specification hierarchy is known after the first traversal of the hierarchy $(a_{kl}, k = 1, ..., N)$, and the maximum allowed size of each circle, inside its parent module with fixed size, is to be determined by calculating the factor, u_l, by which it is to be scaled, without overlapping other circles:

$$u_l = \frac{r_{kl}}{a_{kl}} \quad (3)$$

where r_{kl} $(k = 1, ..., N)$ is the size of circle k, which is to be calculated at node l of the specification hierarchy.

The necessary condition for the circles not to overlap is:

$$r_{il} + r_{jl} = a_{il}.u_l + a_{jl}.u_l \le d_{ij,l} \quad \forall i, j \in 1,...,N \quad (i \ne j)$$

where $d_{ij,l}$ is the real Euclidean distance between points i and j at node l in the specification hierarchy. We define the distance between point pairs (within node l) as $D_{ij} = d_{ij} /(a_i+a_j)$ so that the closest pair algorithm which normally finds $min_{i,j}(D_{ij})$ can be applied to calculate $u = min_{ij}\ (d_{ij} /(a_i+a_j))$ and to find the circle pairs which touch each other first.

After calculating the maximum size for each circle (Figure 10.b) from the force-directed point placement (Figure 10.a), a two-dimensional compaction is performed to reduce the wasted area, in such a way that their relative positions are preserved (Figure 10.c). By enclosing the placed circles with a minimum bounding circle, and scaling appropriately (Figure 10.d), the algorithm determines the placement of circles as well as their sizes within each of the modules at each level of the specification hierarchy.

In this algorithm, the parent module sizes are assumed to be fixed and the maximum submodule sizes are to be determined. This is true for the top-down passes. In the second and the third bottom-up passes, however, the radii of the submodules are known (r_i) and the size of the parent module needs to be calculated. To do this, the same algorithm is used, where an initial size for the parent circle is set arbitrarily, and after determining the actual sizes of the submodules relative to the parent module $r_{i,calc}$, the scaling factor

$$SF = r_i / r_{i,calc} \quad (4)$$

is applied to determine the size of the parent module. Note that the placement algorithm, in the bottom-up passes, takes the r_i's as an input and calculates the radii relative to the parent module size. Therefore, SF is independent of i.

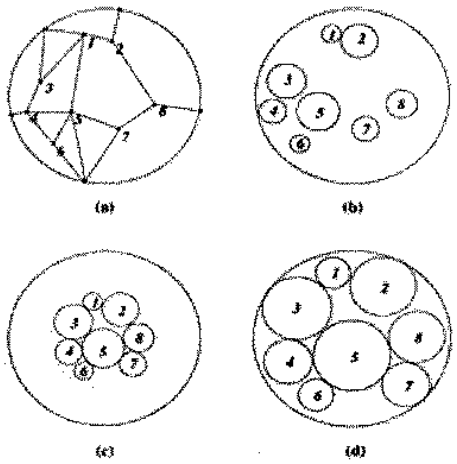

Figure 10. The placement of circles with known relative sizes within a fixed size circle. (a) Point placement by the force-directed method, (b) calculating the maximum allowed size without overlap, (c) compaction to center, and (d) scaling

6. Experimental Results

The framework was implemented in the object-oriented C++ language. Each module is taken as an object containing its own encapsulated data structure and set of functions. Therefore, the functions corresponding to a module apply to the data which are local to that module. For example, a function called , belonging to a particular module, calculates the positions of submodules within that module and encapsulates the module position information in the modules' private data structure. This information does not need to be accessed by other modules' functions. The information which needs to be propagated, such as port position, is made accessible to those modules requiring it.

Due to the lack of hierarchically specified benchmarks for floorplanning and placement, the flat MCNC benchmarks were adopted as test cases and partitioned recursively based on an algorithm proposed in [10] to construct hierarchical specifications. Two of the test cases are macrocell benchmarks called *Ami33* with 33 modules and *Ami49* with 49 modules. These were partitioned into 2 levels of hierarchy (other than the root level) with branching factors of 10 and 16 for Ami33 and Ami49, respectively. The hierarchical test cases produced from the Ami33 and Ami49 benchmarks were

called *Ami33_ha* and *Ami49_ha*, respectively. In order to experiment with a large design, a standard cell benchmark, called *Primary1*, with 833 modules were partitioned into 3 levels of hierarchy with a branching factor of 10. This was called *Primary1_ha*. A summary of the characteristics of the benchmarks (flat specifications) is given in Table 1.

Table 1. MCNC Benchmarks

Benchmark Name	No. of Modules	No. of Nets	No. of IO Ports	No. of Ports
Ami33	33	119	42	257
Ami49	49	408	22	953
Primary 1	833	985	81	3103

6.1 Experiments

The initial framework was applied to each of the hierarchically specified benchmarks. The first four design hierarchy traversals required negligible time (2 sec) compared to the last traversal (9 sec). To improve packing, simulated annealing was used at each node in the specification hierarchy. To preserve the relative placement determined prior to simulated annealing and for efficiency reason, a low starting temperature (10) was set, and a linear cooling schedule ($T_{i+1} = 0.7\ T_i$) and a restricted move set was used, which allowed the submodules to move only within the parent module by a maximum distance defined to be 25% of that module's dimensions (movement window).

6.1.1 The effect of the first four processing passes

The effect of the first four traversals of the design specification hierarchy on the quality of results was observed by deleting these passes from the process and performing the final bottom-up pass without using the information which was normally obtained from these processing passes. Experiments were done using three test cases, with different random movement sequences in the simulated annealing algorithm.

All experiments showed that connection length is decreased when the first four processing passes are performed before the final placement pass is committed. The average amount of saving in the total connection length (half perimeter length of the bounding rectangle), over 20 experiments, was 18.58%. The time spent on the first 4 passes was about 12% of the total process time. Therefore, these passes can help to quickly obtain a good global view of the design prior to the final processing pass.

6.1.2 Orientation optimization process

In the last bottom-up processing pass, for each module in the hierarchy, when the placement of sub-modules is performed, the algorithm allows the parent module to be rotated and/or flipped in order to best meet the port positions of its enclosing circle (set in previous steps). This adjustment reduced the total wiring length on average (for all test cases and different versions of the framework) by 14%.

6.1.3 Branching factor of the hierarchy tree

One of the limitations in hierarchical approaches, such as BEAR 0 and FRODO [11], is the branching factor of the hierarchy tree which is normally no more than 5. Such limitation results in dead spaces in the final layout because the more modules that exist at each node in the hierarchy, the more choices are available in the placement configuration. To see the amount of dead space which may be created in two simple examples, Figure 11 compares the shape function of a module containing 3 submodules with that of a module with 6 submodules where the total area of the two sets of submodules are equal (10). The shape functions are drawn against a constant area curve (that is, height = 10/width) in which no dead space is assumed. The region between the shape functions and the constant area curve is attributable to dead space. The module with 6 submodules tends to be closer to the constant area curve, thereby saving more space than the 3-submodule case. (This fact may force a floorplanner to break the specification hierarchy over one or more levels to increase the number of modules being placed in order to avoid too much dead space.)

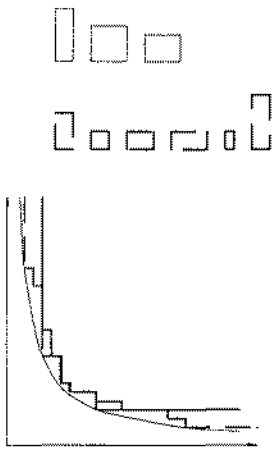

Figure 11. The shape function of a module with three submodules (solid line) and a module with 6 modules (dashed line). The region between the zero dead space curve (thin solid line) and the two shape functions shows the amount of dead space which results for each case

In our framework, unlike the conventional hierarchical floorplanners[11,17], the branching factor of the hierarchy tree is not restricted. Therefore, the dead space problem can be ameliorated. In an experiment, Ami49 benchmark was repartitioned based on a cluster tree generated by the BEAR floorplanning tools which has a branching factor of 4. Our framework was applied to this design and the result was compared with that resulting from Ami49_ha, where the maximum branching factor was 16. The experiment showed that 19.3% of area attributable to dead space is saved when the hierarchy with the larger branching factor is used. Figure 12 and Figure 13 show the results of the application of the framework for the two test cases.

6.1.4 Comparison with ather approaches

The framework presented in this paper is not merely a macrocell placer and floorplanner, since it is intended for hierarchically specified designs where the original

specification hierarchy is to be preserved. However, in order to compare it with other tools, two successful floorplanning/placement tools, TimberWolf-MC (version 6) and BEAR-FP (version 1.1) were run on the test cases. TimberWolf-MC is a macrocell placement tool utilizing simulated annealing [14] and BEAR-FP [17] is a tool which hierarchically reduces the complexity of macrocell placement/floorplanning problems. Neither of these tools was designed to work on an arbitrary specification hierarchy, and furthermore, they work less well on large designs. Another factor which makes the comparison not ideal is the chip aspect ratio constraint which is applied to the above tools but is relaxed in our framework. This author was unsuccessful in attempting to relax this constraint for both comparison tools.

Our framework was applied to the hierarchically specified test cases using the same computer as was used to run the above tools (a system with a 850 MHz AMD Duron and 256 MB RAM). To calculate the routing area around each module, we used the BEAR-FP *quick routing area estimation* approach which is based on the number of ports on each module:

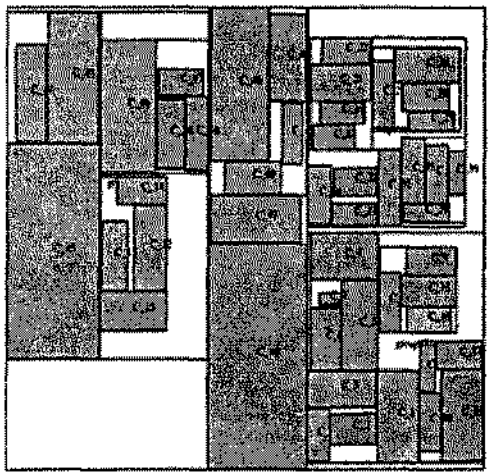

Figure 12. The result of our framework for Ami49 benchmark partitioned with 4 levels of hierarchy and with a branching factor of 4

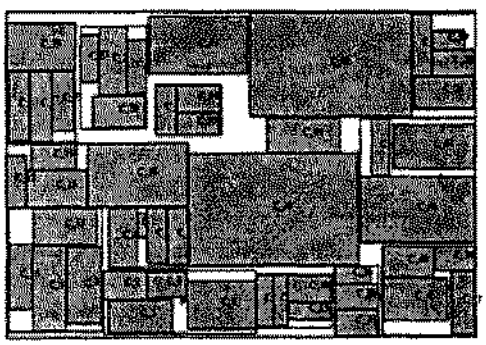

Figure 13. The result of our framework for Ami49_ha test case

$$Routing\_area\_estimation = \frac{(Number\_of\_Ports \times Wire\_Width)}{(2 \times 1.325)} + 4 \times Wire\_Width$$

for the leaf modules and

$$Routing\_area\_estimation = 4 \times Wire\_Width \quad (5)$$

for the parent modules. TimberWolf-MC was applied to the small benchmarks (Ami33 and Ami49) with and without its *fast* option.

Without the fast option, it produces better results than our framework, in terms of total connection length[3], at the cost of very long run time (that is, 573 seconds versus 10 seconds for Ami33, and 1496 seconds versus 30 seconds for Ami49). The chip area produced by TimberWolf was larger than ours (see Figure 14, and Table 2) mainly because the routing area estimation algorithms are different. Nonetheless, TimberWolf, which works on a flat design, may result in designs with significantly less dead space, as discussed in the previous section. However, since TimberWolf did not remove all module overlaps (including the routing area overlaps) in the final result, an overlap removal process will increase the total area of the design. To make the run times more comparable, the fast option of TimberWolf was used with a factor of 10. In this case, the speed of TimberWolf is still about 20 times slower than our framework (see Table 4) and in this mode, it produces inferior results to ours by 11.9% for Ami33 and 10.9% for Ami49 (see Table 3). For Primary1, TimberWolf did not finish the job, even for fast = 20, because of lack of sufficient memory on our computer. Our framework produced a final result in about 4.5 minutes. The final result is shown in Figure 15.

Table 2. Comparison of area (mm^2) produced by the three tools

	Ami33	Ami49	Primary1
TimberWolf	2.42	55.6	Did not finish
TimberWolf (fast = 10)	2.12	59.4	Did not finish
BEAR-FP	1.67	43.1	Did not finish
Our Framework	1.68	44.19	37.66

BEAR-FP was applied to the benchmarks without its final optimization post-processing, namely flat level final compaction and final orientation optimization. The post-processing was not attempted because they can be applied equally to the outputs of our framework and BEAR-FP. In addition, the final compaction procedure may destroy the original hierarchy which is undesirable for our purpose.

The results of our framework, in terms of area and connection length, are very close to the BEAR-FP results (see Table 2 and Table 3). One of the reasons that BEAR-FP produces better results is that it explores almost the whole search space to find the best result. However, for large designs, this strategy may not work, since for Primary1 test case, BEAR-FP was unable to finish the job (it stopped after floorplanning two levels of hierarchy).

6.1.5 Effects of one-dimensional compaction at the top-most hierarchy level

As a post-processing step, the modules at the top-most level of the specification hierarchy can be compacted using two successive one-dimensional (horizontal and vertical) compaction processes. This sometimes saves area but, on the average, results in increasing the total wire length. For example, in the case of Primary1_ha, the final compaction reduced the area of the design by 0.4% but the total wiring length was increased by 4.7%.

Table 3. Comparison of total routing length (half perimeter) (*mm*) produced by the three tools}

	Ami33	Ami49	Primary1
TimberWolf	52,366	865,834	Did not finish
TimberWolf (fast = 10)	81,075	1,192,332	Did not finish
BEAR-FP	66,909	809,376	Did not finish
Our Framework	66,982	1,075,551	2,538,300

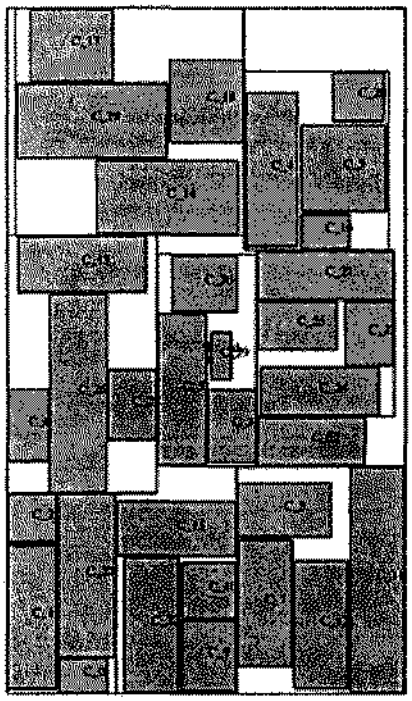

Figure 14. The result of our framework applied to the Ami33_ha test case

7. Conclusions

The framework presented in this paper, performs floorplanning and placement with the objective of preserving the specification hierarchy. This hierarchy can be derived from the structural specification or from a repartitioning process which partitions a flattened design based on such physical constraints as the degree of connectivity between modules. The consideration of inter-dependence among different parts of the design hierarchy is the main characteristic of this framework.

In macrocell designs, due to the random structure of modules and their arbitrary shapes, sizes and port positions, the inter-dependence problem is more critical. The sizes (and therefore the floorplan) of modules at upper levels of the specification hierarchy depend on the placement of lower level modules and the placement of modules at lower hierarchy levels is subject to the determination of the optimal port positions (and the floorplan) of the upper level modules in the hierarchy. Therefore, premature decisions tend to produce poor results. The framework presented in this paper propagates appropriate information to those hierarchy levels in the design which lack the information necessary for performing placement and routing, in a stepwise refinement fashion.

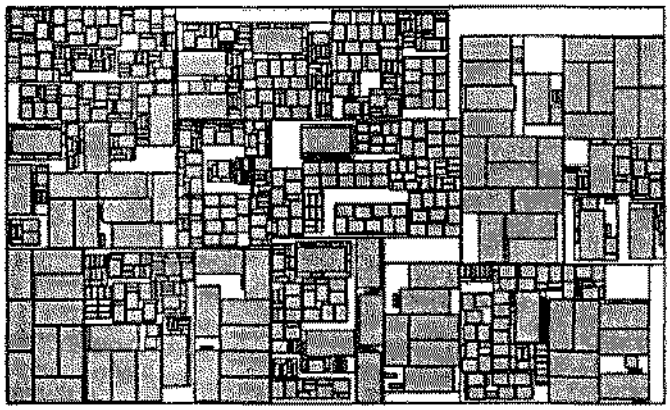

Figure 15. The result of our framework applied to the Primary1_ha test case

Table 4. Comparison of computing time (*seconds*) required by the three tools

	Ami33	Ami49	Primary1
TimberWolf	573	1496	Did not finish
TimberWolf (fast = 10)	165	754	Did not finish
BEAR-FP	19	46	Did not finish
Our Framework	10	30	282

In the high level synthesis process, it is usually desirable to obtain a quick floorplan/placement of the design in order to use physical constraints, such as performance, in the design process. The experimental results showed that this framework can produce good results very quickly for very large designs. While the competitor, placement and floorplanning tools did not finish the job for a relatively large design, this framework produced its results in a few minutes.

The first four traversals of the specification hierarchy which are basically for information gathering proved to be very

quick, but the interconnect length saved as a result of performing these passes proved to be large.

References

[1] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, 2nd ed., Kluwer Academic Publishers, 1999.

[2] A. Kahng, "Classical Floorplanning Harmful," *Proc. Of IEEE Intl Symp. Physical Design*, pp. 207-213, 2000.

[3] P. Chen, and E. S. Kuh, "Floorplan Sizing by Linear Programming Approximation," *Proc. Of IEEE Intl Design Automation Conf.*, pp. 467-471, 2000.

[4] X. Tang, and D. F. Wong, "Floorplanning with Alignment and Performance constraints," *Proc. Of IEEE Intl Design Automation Conf.*, pp. 848-853, 2002.

[5] Y. Feng, D. Mehta, and H. Yang, "Constrained Modern Floorplanning," *Proc. Of IEEE Intl Symp. Physical Design*, pp. 128-135, 2003.

[6] Y. Ma, X. Hong, S. Dong, S. Chen, Y. Cai, C.-K. Cheng, and J. Gu, "An Integrated Floorplanning with an Efficient Buffer Planning Algorithm," *Proc. Of IEEE Intl Symp. Physical Design*, pp. 136-142, 2003.

[7] R. Camposano, "Physical Design: The Whole Enchilada," *Proc. Of IEEE Intl Symp. Physical Design*, pp. 3, 2003.

[8] B. Eschermann, W.-M. Dai, E. S. Kuh and M. Pedram, "Hierarchical Placement for Macrocells: A "meet-in-the-middle" approach," *Proc. Of IEEE Intl Design Automation Conf.*, pp. 460-463, 1988.

[9] M. Pedram, and B. Preas, "A Hierarchical Floorplanning Approach," *Proc. Of Intl Conf. Computer Design*, pp. 332-338, 1990.

[10] W.-M. Dai, and E. S. Kuh, "Simultaneous Floorplanning and Global Routing for Hierarchical Building-block Layout," *IEEE Trans. Computer-Aided Design*, vol. 6, no. 5, pp. 828-837, 1987.

[11] T. Lengauer, R. Mueller, " Robust and Accurate Hierarchical Floorplanning with Integrated Global Wiring," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 6, pp. 802-809, 1993.

[12] B. Schuermann, J. Altmeyer, and G. Zimmermann, "Three-phase Chip Planning — an Improved Top-down Chip Planning Strategy," *Proc. Of IEEE Intl Conf. Computer-Aided Design*, pp. 598-605, 1992.

[13] S. Kirkpatrick, C. Gelatt, M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671-680, 1983.

[14] W. Swartz, and C. Sechen, "New Algorithms for the Placement and Routing of Macro Cells," *Proc. Of IEEE Intl Conf. Computer-Aided Design*, pp. 336-339, 1990.

[15] N. Quinn, "The Placement Problem as Viewed From the Physics of Classic Mechanics," *Proc. Of IEEE Intl Design Automation Conf.*, pp. 173-178, 1975.

[16] F. Preparata, and M. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, 1985.

[17] W.-M. Dai, B. Eschermann, E. S. Kuh, and M. Pedram, "Hierarchical Placement and Floorplanning in BEAR," *IEEE Trans. Computer-Aided Design*, Vol. 8, No. 12, pp. 1335-1349, 1989.

1 Since the hierarchy specified at the behavioral level has been shown to be unsuitable for physical design in many cases, the hierarchy used is assumed to be the structural hierarchy.

2 In case where two points take exactly the same position, they are forced to be separated by a small amount.

3 The total connection length for the results of our framework was calculated on the flattened design for comparison.

Morteza Saheb Zamani received the B.S. degree from Isfahan University of Technology in 1988, and the M.Eng.Sc. and Ph.D. degrees from University of New South Wales in 1991 and 1996, respectively, all in computer engineering. He is a Senior Lecturer in the Department of Computer and IT Engineering at Amirkabir University of Technology. His areas of interest include electronic design automation, VLSI design and reconfigurable computing.
Email: szamani@ce.aut.ac.ir