PLAYOUT - A Hierarchical Layout System

Bernd Schürmann
Gerhard Zimmermann
University of Kaiserslautern
D-67653 Kaiserslautern

1. Introduction

Very Large Scale Integration (VLSI) has become a technology of large economical importance. Mastering of this technology on one hand means the latest semiconductor processing technology, on the other hand computer aided design. Whereas the first is a problem for physics and chemistry, the progress in the second field results from large activities in electrical engineering and computer science. Although a large body of knowledge about solutions to partial problems exists, there is still a lack of combining all these results into integrated systems that are applicable to design very complex VLSI circuits. This is especially nowadays the case where we can see a shift from bottom-up to top-down design.

The PLAYOUT project aims at a solution of the systems problem, based on hierarchy, top-down planning and integrating design tools in a CAD framework which is based on an object-oriented data model and a net-based design flow model. PLAYOUT supports the physical design from behavioral synthesis to the generation of the mask data. PLAYOUT tries to enforce correctness, consistency, and completeness of designs in order to minimize checking requirements. PLAYOUT can be used as a fully automatic layout generator, but sufficient control is given to the designer to interact with the design process. Only the inclusion of designers experience will, in general, result in better than just acceptable layouts. Due to its strong focus on planning, PLAYOUT is also a useful *early design tool* for evaluating system and logic synthesis. The estimation component of physical circuit characteristics is the perfect back-end of the behavioral design phase.

Compared to existing layout tools without a planning component, we hope to achieve better results and less iterations in the design cycle. These advantages could be shown by a large test design of a circuit with nearly 300,000 standard cells using a prototype implementation of the PLAYOUT design system. We will use this example throughout this paper to explain the function of the PLAYOUT system.

This article emphasizes the system approach, the planning aspect, and the use of a hierarchy. New or improved algorithms will be of minor interest. Section 2 will give a short overview of the PLAYOUT design system. Section 3 will then describe our (top-down) design methodology while Section 4 will describe our main toolboxes in more detail. At the end, we will present some aspects of our CAD framework in Section 5 before we will conclude the paper by addressing a large test design in Section 6.

2. System Overview

Figure 1 shows the PLAYOUT system architecture. The system is composed of independent toolboxes which perform the CAD functions. The toolboxes are separated because of different implementation languages and data structures. Most toolboxes have been implemented by our group but several foreign toolboxes like TIMBERWOLF [SeL87] and MIMOLA [Mar90, Zim88.1] have been adapted to the PLAYOUT system.

All toolboxes are loosely integrated in the PLAYOUT CAD framework PLAYFRAME. Each toolbox still has its central data structure and may have several tools that read and modify this structure. Design data are retrieved from the design database PLAYBASE in form of PLIF files and loaded into the local data structure [Sie89][1]. External toolboxes are adapted by format converters so that the central design database sees a PLIF file interface only. For toolboxes for which we have access to the source code, the converter has been integrated into the toolbox. Otherwise the converter is a stand-alone interface.

Currently, each toolbox has its own user interface and controller. Control can also be passed down from the design manager [SpS88]. The design manager keeps track of the design progress and directs the database manager.

All design data are stored and administered by an object oriented non standard database. Each PLIF file is composed of a set of design objects that either represent the inputs or the results of a design step in a toolbox. The database manager composes and partitions these files for every toolbox application. The database stores all relations between complex design objects while the primitive design data are stored in the regular file system. The database has exclusive access to these files.

This structure is very flexible, but still efficient. The set of toolboxes can be easily changed, tools can be added to each toolbox and toolboxes can be executed on different computer types. The communication through files can be carried over any LAN and distributed database implementations are possible. With todays databases the level of complex objects as primitives allows for sufficient response times. Design file composition and decomposition on the database side and interpretation and generation on the toolbox side create overheads that are in the same range as the tool execution times. This disadvantage is compensated by the flexibility which is especially valuable during the tool development and prototype phases. With progress in nonstandard database research this file/complex object principle can be replaced in order to reduce the overhead.

Figure 2 shows the dataflow between the toolboxes in a domain-level plane. Three domains

1. PLIF is a design data exchange format [Sie88] like EDIF [EDI87] and VHDL [HDL88]. In contrast to other exchange formats, PLIF has special language constructs for the physical planning phase.

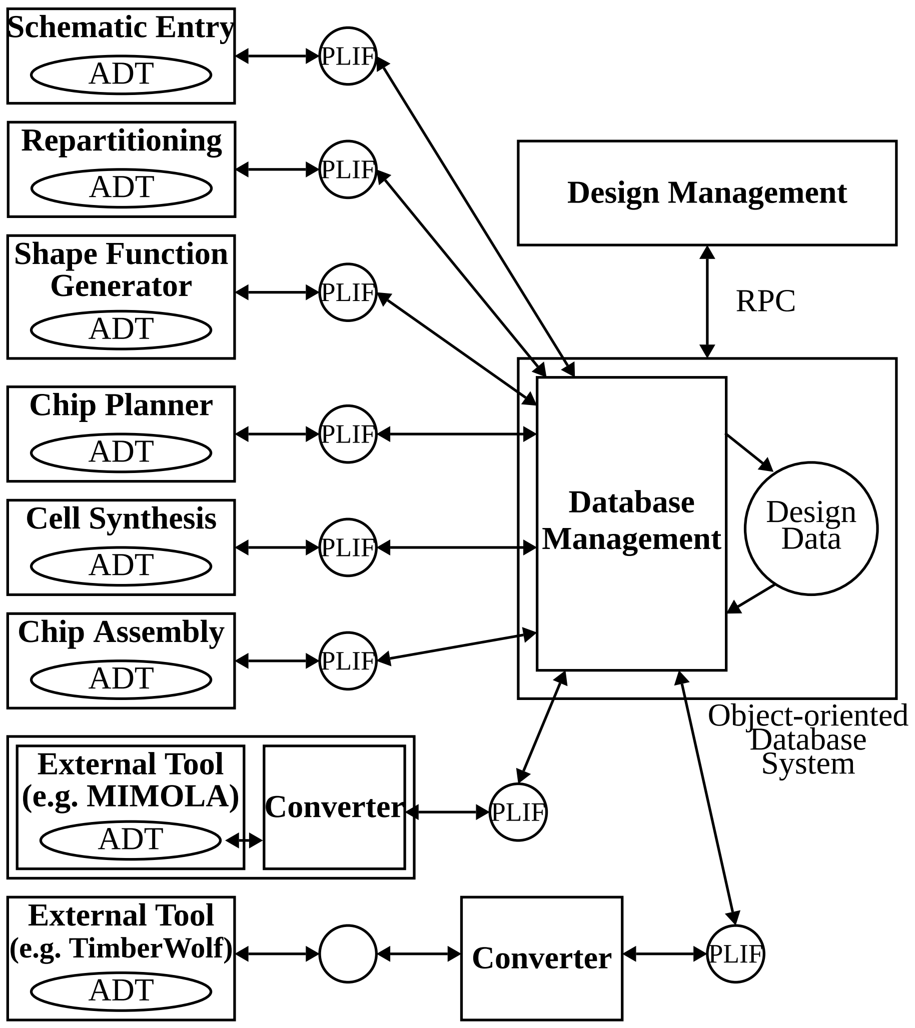


Figure 1 *PLAYOUT design environment.*
All PLAYOUT toolboxes communicate via a design data base and ASCII (PLIF) files. External tools are adapted by converters. The design management is tigthly coupled to the database via remote procedure call interface (RPC).

have been defined earlier [Zim81]: *behavior*, *structure* and the *physical* domain. The early design phases like *Hardware/Software Co-design*, *High-Level Synthesis*, and *Logic Synthesis* cover the left part of the plane while the physical design phase consisting of *Chip Planning*, *Place&Route*, and *Layout Synthesis* correspond to the right side. Verification and extraction steps as well as simulation and test aspects can be found in all design phases. We split the physical domain into *floorplan* and *masklayout* because of our chip planning emphasis.

Cells compose supercells or represent the complete system and are composed of subcells or are leaf cells. This relation defines an aggregation hierarchy which is represented by the levels in Figure 2. The levels can be defined as needed and the three levels in Figure 2 describe an example only.

In this figure, toolboxes, as for example the CHIP PLANNER, are placed on domain boundaries because they transform design data from one domain into another. Synthesis tools transform data of any domain into data of the next domain to the right, and extraction tools work in the opposite direction. Our REPARTITIONER is an alternative within a single domain. All toolboxes

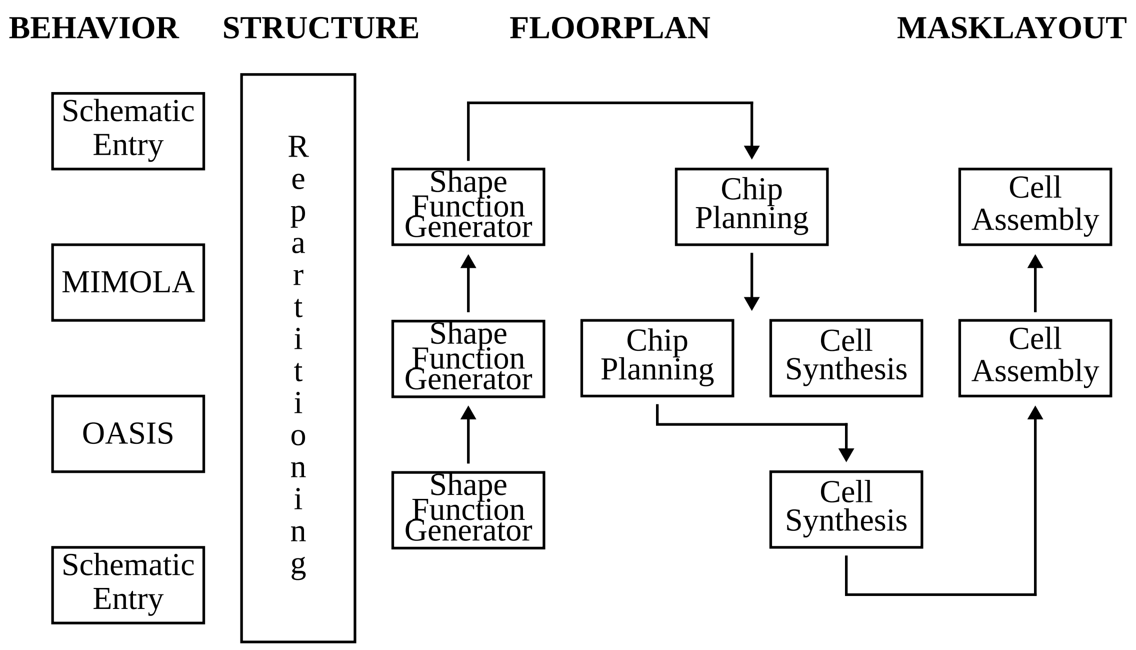


Figure 2 PLAYOUT toolboxes.
This figure shows the most important PLAYOUT synthesis toolboxes. They perform the design over all domains, from behavior up to masklayout. Most toolboxes are used recursively at several hierarchy levels.
The arrows show the general physical design flow.

at least interact with two levels: the cell-under-design (CUD) and its subcells. At the top level the CUD is also called the system-under-design (SUD).

Figure 2 only shows the PLAYOUT tools which are most important for the physical top-down approach. Logic synthesis tools and module generators as well as the test environment are not part of PLAYOUT toolbox set but they are adapted with the OASIS design system [KBK90].

3. Design Methodology

Figure 2 can also be used to explain the design methodology of PLAYOUT. Starting point is a *behavioral description* of the system that has to be designed. This description is normally hierarchical and has to be transformed into a netlist. The synthesis will be performed by different toolboxes depending on the hierarchy level and the type of the behavioral description. The structure may be designed manually and entered via a schematic entry tool. However, this is the most time-consuming and error-prone possibility. More preferable is the usage of high-level synthesis tools at processor level, logic synthesis tools at RT level, and module generators for the basic functions. Structural data of standard functions may also exist in a library. Furthermore, a netlist may already exist from a previous implementation, perhaps with a different technology. Any mixture of sources is possible. A hierarchical netlist is complete if all basic functions are mapped onto layout primitives.

In PLAYOUT we use the high-level synthesis system MIMOLA MSS II [Mar90] that has been developed at the University of Dortmund. For all other circuit descriptions above and below that hierarchy level we used a schematic entry tool for synthesizing our test designs in the past. Fortunately, we now can use the logic synthesis part of the OASIS design system instead of designing all these cells manually. Hardware/software co-design aspects are not considered by PLAYOUT until now.

In our running example that will be addressed in more detail at the end of the paper, 12 processing elements had been described behaviorally and synthesized with MIMOLA. A MIMOLA-TO-PLIF interface that is integrated into MIMOLA generated the netlists at the processor level and a module list at the RT-level. In order to create the internal structure of the RT-modules, we used our schematic entry toolbox. Many of these RT-modules like multiplexers were refined in many hierarchy levels because of their high degree of regularity. At the lowest level the basic functions were mapped onto the commercial Siemens standard cell library ACMOS4H. At the upper-most hierarchy level, the twelve processors were connected to implement an asynchronous computing system by again using our schematic entry tool. We called the top-level circuit XLII.

The structure hierarchy normally shows partitions that resemble functional units as for example processors, adders, flip-flops, or gates. Rarely this partitioning is good for physical design. Physical design tools generate good results only if the number of subcells and the relative sizes of the subcells are within a certain range. Both conditions are violated by typical behavioral designs. In most cases, we have too many register transfer blocks as part of a processor netlist which may be large memories as well as small inverters. On the other hand, the register transfer blocks are built up by a deep hierarchy with few modules at each level. The cells at system level also consist of a small number of subcells only.

We call the modification of the behavior-oriented netlist hierarchy into a new hierarchy which fits to the physical design phase *repartitioning*. The automation of repartitioning is still an unsolved problem in VLSI design. A “flat” design with only one level of hierarchy would not solve the problem, because the complexities of the resulting placement and routing problems are too high to be feasible. Flattening the hierarchical netlist and then computing a new hierarchy from scratch (which is called *partitioning*) is neither a good approach because we then loose the whole regularity of the circuit. The modification of the hierarchy has to be as small as possible.

After repartitioning, hierarchical layout is generally done bottom-up. This means that first the layout primitives are combined to form detailed layouts of larger cells, these are then combined again until the chip is complete. At each level optimal layouts are sought, resulting in corresponding shapes and pin positions of each cell before it is used at the next higher level.

The problem of the bottom-up approach is that local decisions govern global ones. It is unlikely that different shapes (rectangles or more complex polygons) will fit together without wasted area. If the placement of subcells is optimized for high area utilization, wiring length and routing area at the higher hierarchy level will increase, as already has been observed by Breuer [Bre83].

Top-down chip planning tries to avoid wasted area and large signal delay times on higher (global) levels. Shapes are planned which will fit together with minimal dead area and which can be placed with minimal wiring length and total area at the higher levels. In order not to push the waste to the lower levels, a prediction of the influence of the global placement on the local subcell area and timing behavior is necessary. Specifically, this means a prediction of the influence of shape and pin locations on the characteristics of the subcells. The influence on the subcell area, which is already well examined, is generally expressed by shape functions. In contrast to that, the influence on the subcell's timing behavior is still an important research topic. The goal of chip planning is a trade-off between global and local aspects in order to find better layouts than with purely bottom-up methods.

After repartitioning our area estimator called SHAPE FUNCTION GENERATOR calculates shape functions for all cells in the hierarchy. Inputs are the known shape functions of the used cell library (leaf cells) and the complete structural description. Figure 2 shows the bottom-up data-flow of three levels of shape function generation. Shape functions could be calculated in post-order in the hierarchy tree. This would cause many redundant calculations in the case of regularity. Redundancy is avoided by exploiting the cell type concept, i.e. by generating the shape functions for all non-leaf *cells* instead of the *instances* in the hierarchy tree. This has to be done in reverse topological order of the cell hierarchy (which can be extracted from the instance hierarchy). The chip level shape function is a prediction of the total chip area. If it fulfills the requirements, a suitable chip shape can be selected. Otherwise iterations through structural design and repartitioning may be necessary. This is a typical application of the SHAPE FUNCTIONS GENERATOR as an "early design tool".

The CHIP PLANNER starts with a predicted shape of the chip and, if given, the pad positions. It tries to place the subcells in a slicing geometry [SzO80] such that the total area including inter-subcell-wiring becomes a minimum. The subcell shape functions and other constraints like maximum length of critical paths are taken into account. The CHIP PLANNER generates a floorplan consisting of the subcell placement and the assignment of the nets to wiring areas. No detailed routing is performed.

For each subcell the planned shape in the floorplan and the approximate pin positions (pin cost functions) are propagated to the next lower level. The CHIP PLANNER also generates a new CUD shape function, now based on the known placement and global routing. This shape function is more precise than the one generated by the SHAPE FUNCTION GENERATOR. It can be used for an adjustment step at the next higher hierarchy level.

The CHIP PLANNER is recursively called at the lower levels for all subcells again. The hierarchy tree can be traversed in breadth-first or depth-first manner. In case of a breadth-first traversal we have several choices. We can either plan all subcells independent of each other (sequentially or in parallel), or we can adjust the floorplan of the CUD after planning each subcell. This iterative planning/adjustment approach allows for a better response to deviations between planned and refined shapes and pin positions of the subcells. Out iterative approach is called *Three-Phase Chip Planning*. However, this reduces the degree of parallelism and there still remains an ordering problem.

Better results are conceivable if all subcells of a CUD are planned in parallel with dynamical update of the CUD floorplan and communication between the neighbor subcells about shape and pin locations. This approach would probably generate the best results and could be distributed on parallel hardware. Still research is necessary to find the right communication parameters and control points and to study the convergence. One of the first parallel planning methods is described in [GlZ92].

In Figure 2 two planning levels are shown. Either more or fewer levels are possible, depending on the complexity of the total structure and the complexity which each toolbox can handle efficiently. The complexity is measured by the number of primitives that have to be placed in total or at each level. The PLAYOUT CHIP PLANNER works best with 30 to 80 subcells.

After chip planning the final mask layout has to be computed bottom-up with respect to the top-down computed floorplans. This phase is called *Chip Assembly* and it is split into *Cell Synthesis* and the following *Cell Assembly.* Cell synthesis is the link between the top-down planning phase and the bottom-up assembly phase. The input for cell synthesis is a frame and pin positions and is similar to the chip planning input. However, the cell synthesis step uses layout primitives (e.g. standard cells) and computes a final layout similar to cell assembly. In PLAYOUT cell synthesis is currently restricted to memories and standard cells blocks to reduce the number of toolboxes to be supported.

Cell assembly uses the floorplan of the CUD and layouts of the subcells as input and computes the final layout of the CUD. Since we already performed a (top-down) planning step for the CUD, no placement has to be done as it is the case for pure bottom-up physical design. In our case, we take over the topology from the floorplan and only adjust the geometry with respect to the subcell layouts. The global routing information of the floorplan is refined by detailed routing. After successful detailed routing the mask layout is complete.

The hierarchy tree need not be balanced. We therefore may find cell synthesis and cell assembly steps at the same hierarchy level.

4. Toolboxes

After discussing the global design methodology we will now go into some more details of the toolboxes. Although PLAYOUT supports the whole VLSI design from behavior to layout, our main focus is the physical top-down chip planning phase. We will therefore concentrate on the toolboxes which cover this part of the design. These will especially be our area estimator called SHAPE FUNCTION GENERATOR and the PLAYOUT CHIP PLANNER. Besides these two toolboxes we will also describe the cell assembly based on the floorplan provided by the CHIP PLANNER and we will spend some place for addressing the repartitioning step. The latter is important for preparing the hierarchical netlist for the physical design phase.

The PLAYOUT front-end toolboxes which synthesize the netlist data from the behavioral description of a circuit have already been mentioned in the previous section. For this purpose we mainly use the MIMOLA synthesis system MSII from University of Dortmund and our own schematic entry toolbox for the manual synthesis steps. Recently, we also have access to the logic synthesis component of the OASIS design system from MCNC. Since both synthesis toolboxes have not been developed by our group and both do not belong to the main topic of this contribution we will not go into more detail of these design aspects.

For the remaining part of this contribution we assume that a hierarchical netlist is available no matter how it has been generated. However, the netlist hierarchy reflects the behavioral design. It therefore must be repartitioned to meet the requirements of the layout tools.

4.1 Repartitioning

The goal in repartitioning is the generation of a suitable hierarchy for the physical design phase. A physical hierarchy has some hard and some soft conditions that have to be fulfilled. In this discussion we include levels beyond the chip level because very complex systems may require cabinets, racks, printed circuit boards (PCBs), multi chip carriers and chips.

Each of these levels has different requirements. Examples for hard conditions are the maximum number of slots in a rack, pins on PCB connectors, or of pads on a chip. Total power dissipation and chip area are other hard constraints.

On the chip soft conditions dominate. Currently we try to enforce the following:

- The number of cells placed by chip planning should be between 30 and 80. For a smaller number the flexibility may be too small to achieve dense placements. For larger numbers execution time would become too large for interactive design styles. For cell synthesis using standard cells the number should be between several hundreds and several thousands for similar reasons.
- The subcells of a CUD should not differ in area by more than a factor of 10. Especially for slicing geometries it is hardly possible to place very different cells together in a slice without stretching the smaller one beyond reasonable aspect ratios. On the other hand only few good placements exist for cells of equal shape. Thus a good distribution of subcell sizes in each CUD is an advantage.
- Functional hierarchies often display a high regularity by the use of standard functions. The RT-level, the gate level, and complex gate level are good examples. If we would repartition only with both objectives described above, most of this regularity would disappear and no standard function would remain on intermediate levels of the hierarchy. Loss of regularity results in a larger design task. The distribution of standard functions over many partitions would reduce the capability of manual inspection and comprehension of a layout. This would also reduce the possibilities of meaningful manual interactions and improvements.
- The number of nets between partitions should be minimized to the extent of what a good min-cut algorithm would achieve. This is necessary to reduce the overall wir-

ing length by pushing the dense wiring into cells at low levels of the hierarchy and thus small dimension.

- Considering timing conditions is very important. Experiments have shown that the structure hierarchy has a large impact on the timing behavior of the layout. Time-critical nets should therefore be limited to small cells, too. This should not only be achieved on the average but individually. One critical net with a long wiring length may determine the total systems performance.

As we mentioned above, there are two alternative approaches to solve the problem (see Figure 3). The brute-force approach is to flatten the whole hierarchy and then to construct a new hierarchy from scratch by recursively applying an n-way partitioning algorithm. This method is depicted by the lower part of Figure 3.

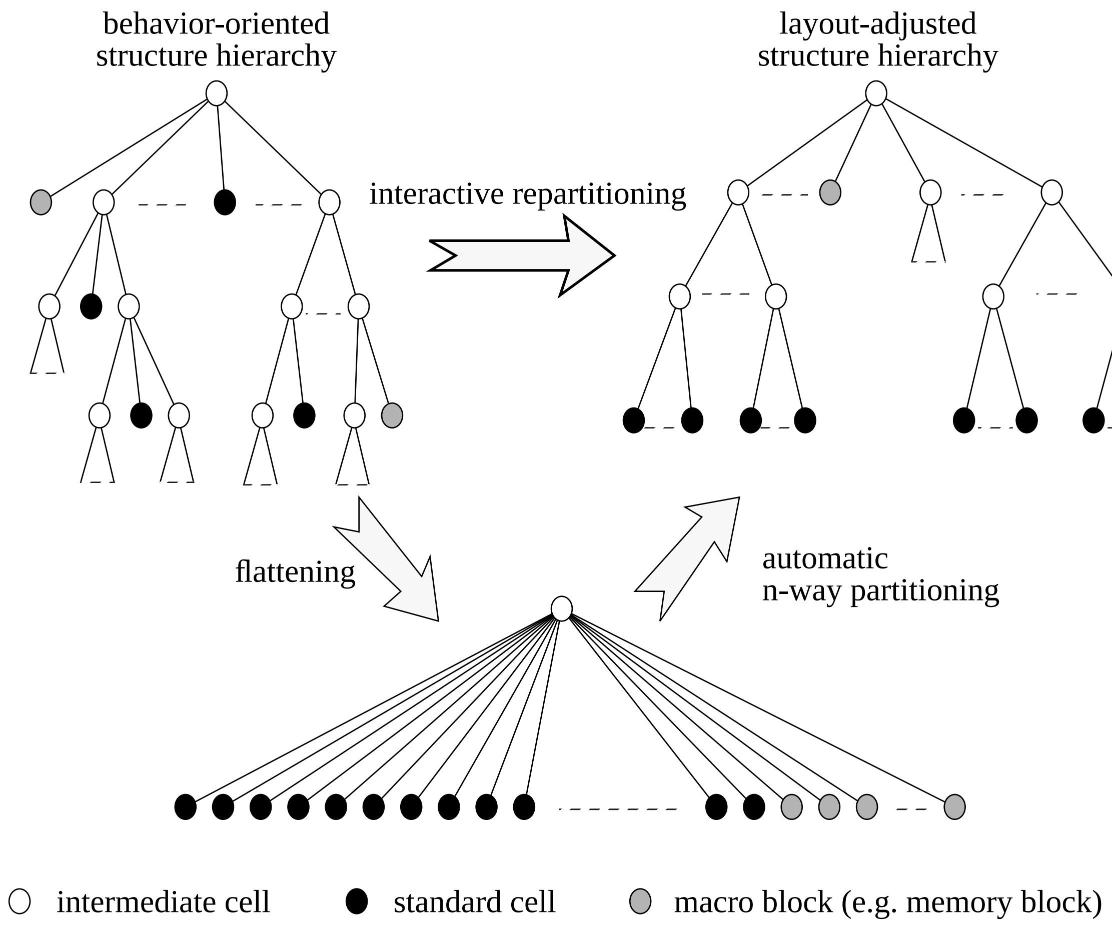

behavior-oriented structure hierarchy
layout-adjusted structure hierarchy
interactive repartitioning
flattening
automatic n-way partitioning
intermediate cell
standard cell
macro block (e.g. memory block)

Figure 3 Alternative repartitioning approaches

The problem for this approach is the large number of different objectives as described above. This number of conditions can be easily extended and it is difficult to merge all of them into a single cost function that would be necessary for the partitioning algorithm. Our idea therefore is to achieve the goals by small, local changes to the original hierarchy using an interactive repartitioning tool.

Opposite to the automatic n-way partitioning method would be a manual repartition approach. The repartitioning tool would only keep track of a consistent hierarchical netlist but all decisions have to be made by the designer. A more sophisticated approach would be extending the interactive tool by analysis and proposal functions.

The development of the PLAYOUT REPARTITIONER followed the latter idea. In order to gain experiences, we first implemented an interactive repartitioning toolbox without any sophisticated functionality. A graphical user interface supports the designer in performing basic repartitioning steps. For example, we can generate a new cell at any level of the hierarchy (i.e. a new node in the hierarchy tree), replacing a set of selected cells. These cells become subcells of the new cell and thus move down in the hierarchy. Delete is the reverse operation. We can also move cells from one location in the tree or graph to another. The distinction between tree and graph operations is the difference between changing only one instance and the type of a cell, respectively. In the latter case a change affects all instances of a cell (type). From these basic functions all possible changes of the tree or graph can be constructed. All functions are controlled by menus.

This basic repartitioning approach was good for first experiments only. The results were often better than automatic partitioning because we could keep the regularity and other objectives. However, it was not possible to do large changes and to consider timing problems without having analysis tools. We extended the PLAYOUT REPARTITIONER by such analysis functions but also by tools implementing simple strategies like “flattening of all blocks with less than 500 standard cells” [HNZ93]. Various (n-way) partitioning tools considering different objective functions complete the tool set. They are only used to suggest changes but no change will be done without designer interaction.

The current architecture of our REPARTITIONER has similarities with an expert system approach. The analysis and the suggestion tools can be regarded as experts which provide their results via the graphical, interactive interface to designer for decision making. This scenario is very similar to the AI rule-based blackboard architecture [HaR85] which is an interesting approach for our problem. The kernel of the REPARTITIONER corresponds to the blackboard. Additional “experts” can easily be added. The challenge of our research will be replacing the designer step by step with more automatic control instances. Nevertheless, we believe that it will not be possible to fully replace the designer within the near future. Figure 4 shows a simplified view of the blackboard approach.

4.2 Area Estimation

The most important part of top-down VLSI design is (top-down) chip planning. As described above, this step is divided into a bottom-up area estimation and a following top-down planning phase. Figure 5 shows the floorplan section at one hierarchy level. The left part of this figure illustrates the area estimation. Beginning at the level of the leaf cells we compute shape functions for the cells at the higher hierarchy levels. The input for an estimation step are the shape functions of the subcells and the netlist of the CUD. The output curve will be input to the area estimation at the next higher level.

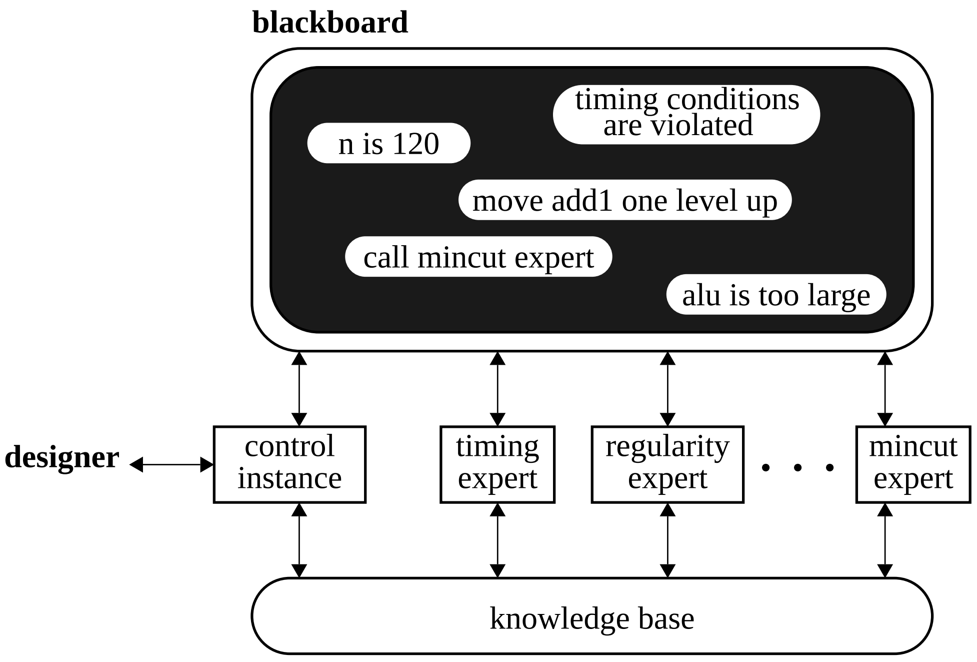


Figure 4 Simplified view of a blackboard architecture for repartitioning

The shape functions are one of the most important inputs for chip planning. The planner uses the curves of the subcells and the netlist of the current cell for computing a floorplan as small as possible. For top-down chip planning, we have a further input. This is a frame with fixed dimensions and pin intervals at the border. It is a result of the preceding planning step at the next higher hierarchy level and it is based on the shape function of the current cell. The resulting floorplan should match this frame.

The output of a planning step is the floorplan that will be used by the cell assembly and the frames of the subcells which are input at the next lower hierarchy level. However, the quality of chip planning depends to a large extent on the quality of area prediction which is modeled by shape functions. In order to understand the area estimation we take a short digression into our geometric model.

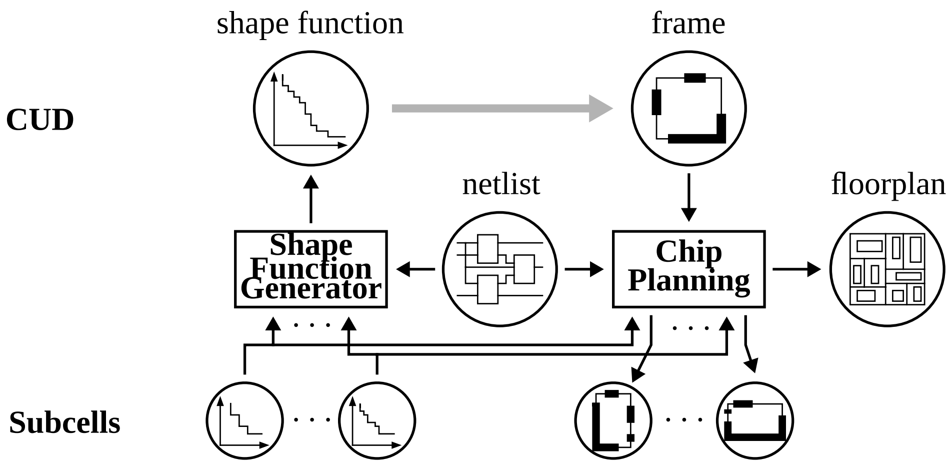
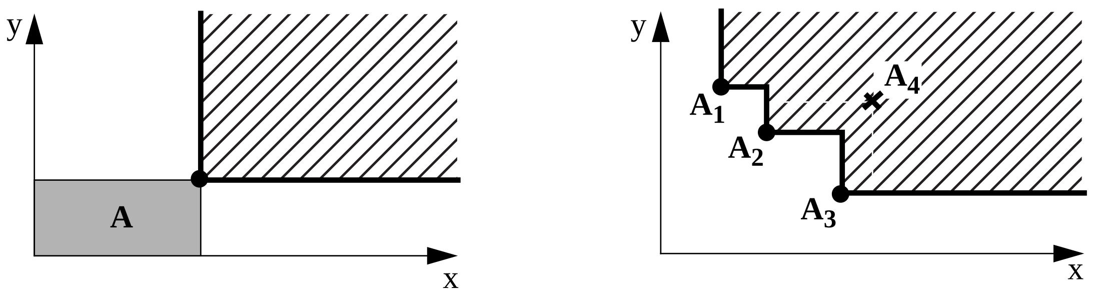


Figure 5 Interaction between aArea estimation and chip planning

4.2.1 Shape Functions

Our goal is the estimation of shape functions of rectangular layouts. Shape functions form the border between feasible and infeasible shapes. Figure 6a shows the shape function of a fixed macro cell *A* for which only one layout alternative exists. The black dot (corner) defines the x and y dimensions of the rectangular cell *A*. All other points on the curve and in the hatched area are made feasible by adding empty space to *A* on any of the four sides or within *A*. It can be used for wiring.



a) one layout alternative *b) four layout alternatives*

Figure 6 Shape function of a macro cell

IF *A* has different shapes (for example A_1, ..., A_4) by controlling the layout of *A* with different parameters, then a shape function as in Figure 6b evolves. Per definition as a lower area bound, the shape function is monotonic and A_4 could be constructed by adding empty area to A_2 or A_3. The shape function in Figure 6b is therefore fully defined by the corner coordinates A_1, A_2, A_3.

A_4 may have properties that neither A_2 or A_3 have. For example, A_4 may have all pins on one side, whereas A_2 and A_3 have pins on all four sides. Thus, in a specific floorplan, A_4 may, despite larger area, produce a smaller floorplan or a shorter critical delay path. Therefore, we do not always enforce monotony.

For flexible cells, only estimated shape functions can exist, if the layout style offers design decisions that influence the layout area. Thus each corner point has tolerances in x and y and the shape function has a tolerance band around the average curve as shown in Figure 7. In the extreme the shape function can be a continuous curve, but due to tolerances and design style restrictions it can always be approximated by a suitable number of corners. Module generators can be either modeled by macro shape functions if they generate fully predictable layouts or by flexible cell shape functions otherwise.

Our basic geometry is slicing. Slicing is the dominant approach in floorplanning because of features which simplify placement and routing. It is not obvious that more complex structures have major advantages. A possible extension to slicing is the deviation of the slicing lines from straight lines.

Slicing dissects the rectangle of a CUD into non overlapping rectangles representing the subcells. The walls between the subcells are the slicing or cut lines. Slicing geometries can be

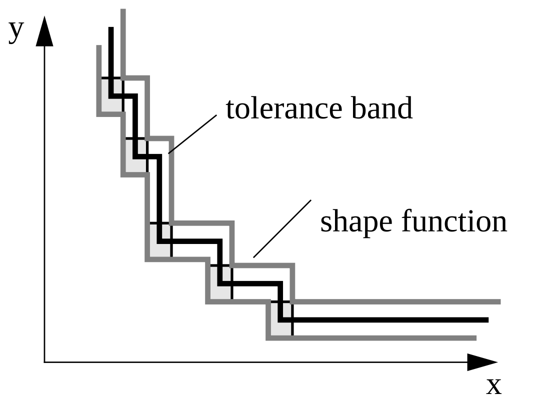


Figure 7 Shape function of a flexible cell

represented by slicing trees. If shape functions for the leaves of a tree are known and the orientations of all slices in the tree are determined, the shape functions can be easily added up the tree and thus for the root node which represents the CUD [Ott83]. Figure 8 shows this correspondence.

If the orientations of the slices are not known, optimal orientations can be determined by adding monotonic shape functions horizontally and vertically and selecting the corners with minimal area of both results (see below).

The resulting shape function estimates the total area only if no additional wiring space is required. We extended the model by trying to estimate this space [Zim88]. This led to a *five color model*.

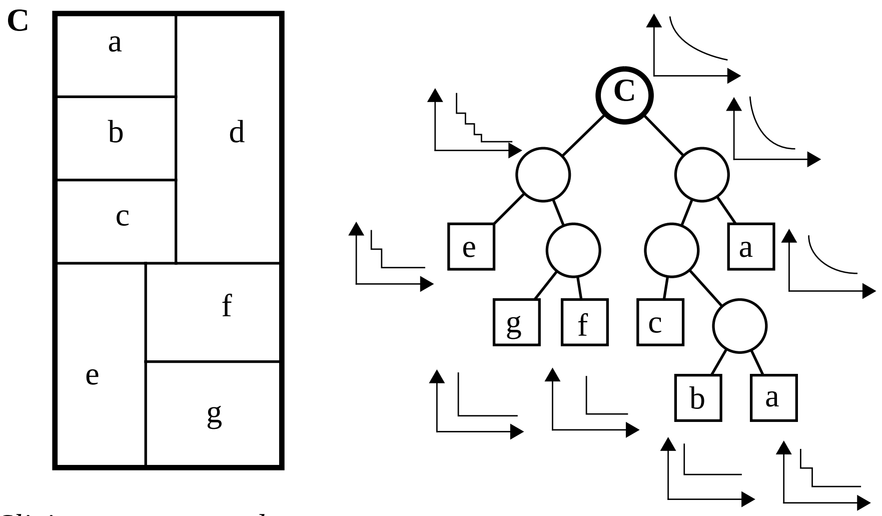


Figure 8 Slicing structure and tree

4.2.2 The Five Color Model

Let us look at any one level of a multi-level hierarchy and let us assume that we seek the shape function of the CUD. The CUD consists of n subcells sub_j, $j = 1 .. n$. Each cell has pins at its perimeter. Nets are globally defined for all hierarchy levels. A net can therefore exist at many levels. For the purpose of estimation we divide nets into segments that are fully contained in one level only.

Such a net segment can be further divided into pink and red parts (Figure 9). The pink part of the net connects only the pins of the subcells sub_j but not the pins of the CUD (dotted lines). Pink segments are internal connections of a cell and exist if more than one internal pin exists. Red net segments connect pink segments (or the only internal pin of a net) to an external CUD pin (solid lines).

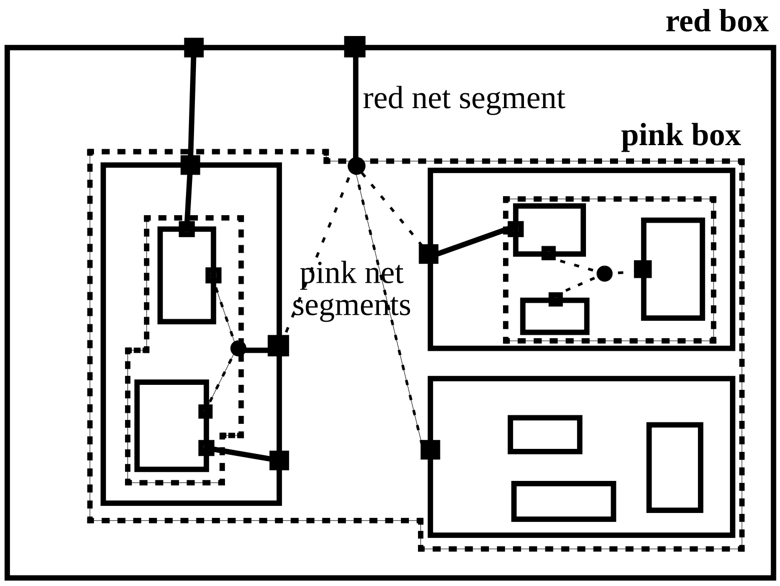


Figure 9 *Pink and red net segments.*
The figure shows net segments on two hierarchy levels.

With this notion of net colors we define wiring area w^{pink} and w^{red} for the CUD. w^{pink} is the sum of all areas occupied by internal wiring and w^{red} is the area necessary for connecting the internal wiring with the CUD border. Accordingly we define enclosing rectangles for both colors, called the pink and the red box, resp. (Figure 9). The areas of both boxes are denoted a^{pink} and a^{red}, resp. In addition, we define a blue box with area a^{blue} for the case that all nets of the current level are disregarded. a^{blue} is the sum of all subcell areas. Note, that the blue and the pink boxes are fictive boundaries because neither all subcells nor the pink net segments are located at a particular part of the chip die without contact to red net segments. Nevertheless, we need the sizes of these fictive boxes for area estimation.

Besides these three colors pink, red, and blue, we use a fourth color black which denotes the empty space (wasted area). This is the area which is neither occupied by the subcell nor by wiring. Since the empty space is spread over the whole CUD, no concrete black box can be extracted. We only use the total amount of empty area for the estimation. This value is denoted by a^{empty} or a^{black}.

In our estimation model, the different colored areas are computed as follows:

- The area of the (fictive) blue box is the sum of all subcells[2]:

$$a^{blue} = \sum_j a^{red}_{sub_j}. \tag{1}$$

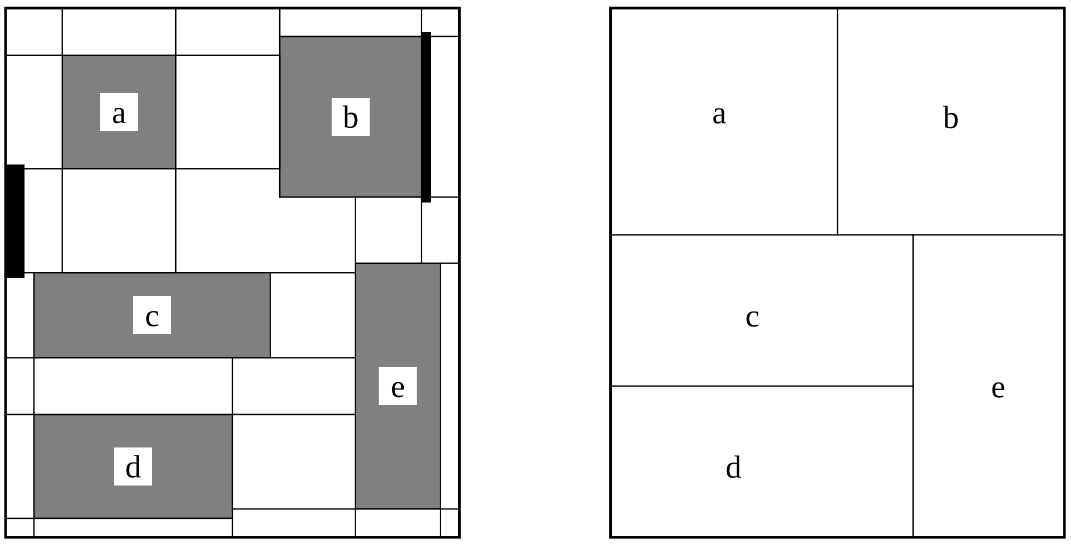
- The pink area includes the blue box (i.e. the subcell area), the area occupied by the pink wiring segments, and the empty space:

$$a^{pink} = a^{blue} + w^{pink} + a^{black}. \tag{2}$$

- The red area contains the pink area and the area of the red wiring segments connecting the internal wiring with the CUD boundary:

$$a^{red} = a^{pink} + w^{red}. \tag{3}$$

For performing floorplanning, these four colors are not enough. We further have to consider the wiring external to the CUD which we denote by the color green. Figure 10 depicts this aspect. Figure 10a shows a floorplan with five subcells *a*, *b*, *c*, *d*, and *e* (gray blocks). We assume the wiring to be located in channels between these blocks. The white rectangles describe channels and the white polygons connecting the channels describe switchboxes. The black boxes model the empty space.



a) red subcells and wiring channels *b) assigning wiring area to the subcells*

Figure 10 Floorplan with five subcells

For computing a correct geometry of the floorplan we have to assign fitting shapes to the subcells. This is done by adding up the shape functions of the subcells and then distributing the CUD area to the subcells with respect to this shape functions (see Section 4.3). For that purpose, the subcell shape functions must include all wiring area. We assign the channel and

2. Corresponding to our color model described so far, the boundaries of the subcells are equal to their red boxes. They include all sub-sub-cells (sub-2-cells) as well as the whole internal (pink and red) wiring area and empty space. With that, layout frames are red, too.

switchbox areas to the adjacent subcells which results in a slicing topology as depicted by Figure 10b. The five rectangles now include the (red) subcell areas and the inter-cell wiring w^{green}. We assign the color green to these rectangles:

$$a^{green} = a^{red} + w^{green} \tag{4}$$

We need this notion during floorplanning of the CUD because w^{pink} and w^{red} are useless parameters for the placement of the subcells:

$$w^{pink}_{CUD} + w^{red}_{CUD} = \sum_j w^{green}_{sub_j} . \tag{5}$$

4.2.3 Shape Function Arithmetic

The input to the shape function computation are the shape functions of the subcells and the netlist of the CUD. Output should be the shape function of the CUD. Since we restrict to slicing structures, we first compute a binary slicing tree by a partitioning step (see Figure 8). We compute the shape functions of the inner nodes of this tree bottom-up until we get the curve of the root node (CUD). The task of each iteration step is to combine the shape functions of two sibling nodes in the tree to get the shape function of their common father node as result.

<u>a) Adding two shape functions</u>

Let us assume that we have two sibling nodes *A* and *B* in the slicing tree which have a common father node. For the example we further assume that both nodes are leaf nodes. However, the procedure described below works identically for all other nodes in the tree. For both nodes we have three different layout alternatives. Figure 11a shows these alternatives and the corresponding shape functions.

We now can place cell *A* beside cell *B* or we can place *A* above or below *B*. Depending on the orientation of the slicing line, we get two different shape function for the common father node. The vertical cut, this means cell *A* is placed beside cell *B*, results in the curve of Figure 11b and a horizontal cut results in the curve of Figure 11c. In all cases, the bounding box of the father node contains more or less empty space which is the result of the slicing topology.

Both curves have to be combined to get a single shape function for the father node. The resulting shape function must again describe the lower bound of all feasible shapes. So, the unification procedure takes those curve segments from *curve 1* and *curve 2* which form this lower bound (Figure 12). In our example, the result is the curve which is shown by the solid black and gray line. Each corner point gets a tag that denotes the cut orientation with minimal area consumption. We call this: *optimal orientation*.

Until now, we considered only the subcell areas which resulted in a correct shape function for the common father node. On the other hand, no wiring space has been considered. This wiring space widens the cell area in horizontal and vertical directions. The combined shape function has to be shifted to the top and the right. This shifted curve would be input to the shape function computation at the next higher level.

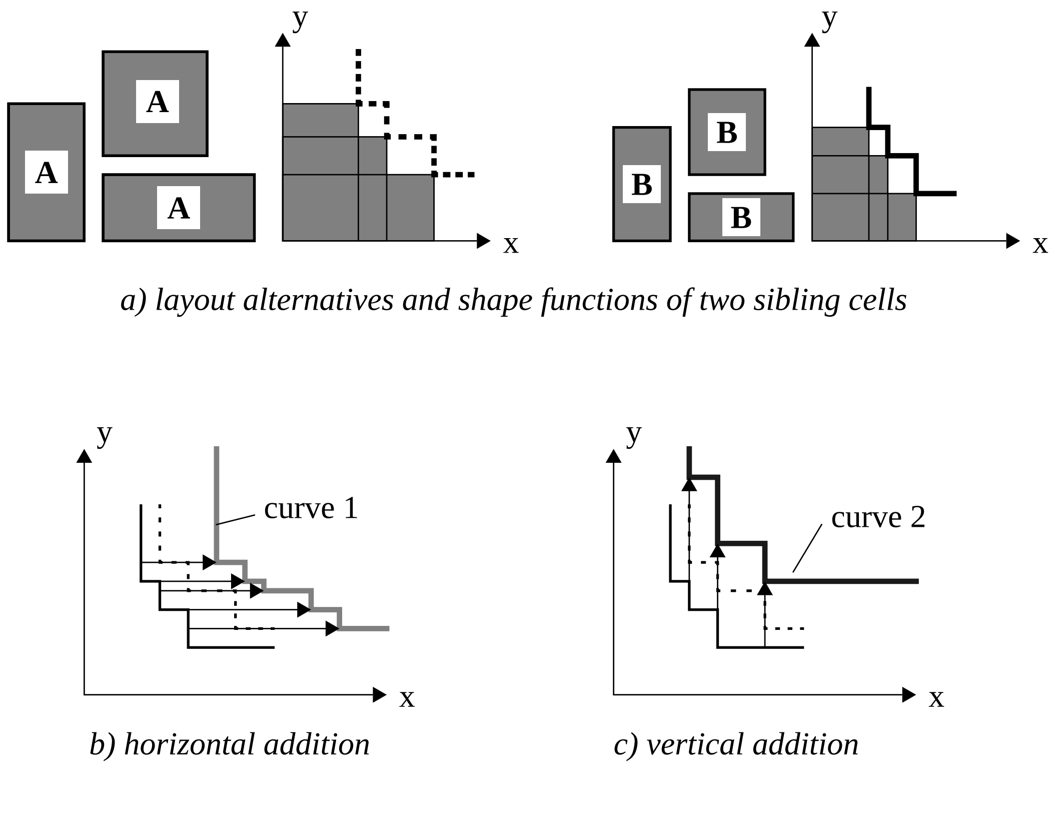


Figure 11 Horizontal and vertical addition of two shape functions

Wiring area can be divided into the pink *internal wiring* and the red *external wiring*. For combining two sibling nodes, the pink net segments of the father node are segments which have connections to both sibling nodes only. They cross the corresponding slicing line. Red net segments have connections to at least one of the two child nodes and any node outside the considered subtree. They may be pink net segments of an ancestor node.

This distinction between pink (→ internal) and red (→ external) wiring is necessary because the area needed for external wiring depends very much on the positions of the pins on the cell border which are not known during the bottom-up area estimation phase. Our basic area estimation model is therefore based on pink wiring only that can be very well estimated without geometrical informations.

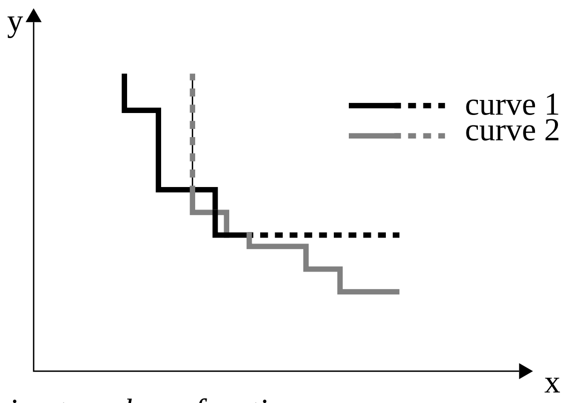


Figure 12 Combining two shape functions

Independent of the shape of a cell, we estimate the area needed to connect its subcells, i.e. the increase of the cell area due to the pink wiring, by the following simple functions (Figure 15):

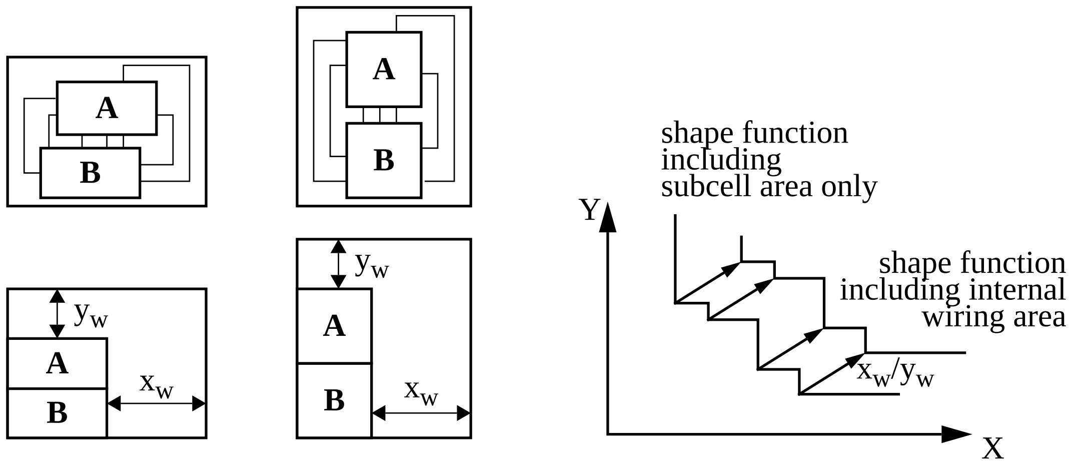

A
B
A
B
y_w
A
B
x_w
y_w
A
B
x_w
shape function including subcell area only
Y
shape function including internal wiring area
x_w/y_w
X

Figure 13 Wiring Area Estimation. The shift of the shape function is independent of the shapes.

$$x_w = n_{pink} \cdot t_x \cdot \alpha_x \quad (6)$$
$$y_w = n_{pink} \cdot t_y \cdot \alpha_y \quad (7)$$

Wires between the subcells need additional space in x and y direction, called x_w and y_w. The horizontal enlargement x_w of the cells is estimated by multiplying the number of internal wires connecting the subcells (n_{pink}) with a statistical track demand factor t_x and a technology dependent scale factor α_x. t_x determines how many vertical tracks a single wire needs on the average. It models the quality of the layout tools and routers. The value is typically about 0.6 which means: one wire needs a little bit more than a half track on the average. α_x is a scale factor describing the width of a single track. It depends on the design rules. The vertical enlargement y_w is estimated accordingly. In total, we have a constant shift of the shape function to the top and right (right side of Figure 15).

b)Feedthroughs

A still open problem is the empty space (wasted area) that occurs due to the corners in the shape functions. If we set two cells on top of each other as shown in figure 14, the composite cell gets the maximum of both x dimensions and the sum of the y dimensions. The difference in x results in empty space a^{empty}.

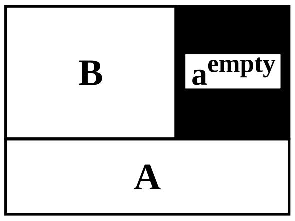


Figure 14 Empty space

In our model, we assume that empty space increases the transparency of a cell. Transparency is modeled by vertical and horizontal feedthroughs. Standard cells, for example, have built-in feedthroughs perpendicular to the row. We measure vertical feedthroughs by the wiring space *xf* in the x-dimension of a cell. The empty space a^{empty} of cell B in figure 14 is modeled as wiring space x^e and thus

$$\hat{xf}_B = xf_B + x^e \ . \tag{8}$$

$\hat{x}$ denotes an area with empty space added. The question now is: What is the vertical transparency of the combined cell *AB*. We use the weighted sum of two extremes. One extreme takes the average of the sum of the transparencies of *A* and *B*, the other the minimum. This results in

$$xf_{AB} = c \cdot \frac{\hat{xf}_A + \hat{xf}_B}{2} + (1-c) \cdot min(\hat{xf}_A, \hat{xf}_B) \quad . \tag{9}$$

A large set of measurements with our design tools has shown that xf_{AB} is much nearer to the minimum than to the average. We set c << 1. The horizontal transparency in this example is easily computed, because no empty space has to be considered:

$$yfc_{AB} = yfc_A + yfc_B \ . \tag{10}$$

The case of cells with a vertical slicing line is handled accordingly. In the calculation of wiring space the feedthrough dimensions are subtracted from the needed wiring space for nets. Any remaining feedthrough space is propagated up in the slicing tree.

4.2.4 Hierarchical Models

Until now, we considered the area estimation on one hierarchy level. We construct a binary slicing tree and add the shape functions of the subcells up in the tree until we get the shape function of the CUD. Wiring area, empty space, and transparencies are considered as described in Equation (6) to (10).

The hierarchical bottom-up area estimation over several hierarchy levels can be done in the same manner. We (virtually) combine the slicing trees of all hierarchy levels into one large slicing tree (Figure 15). In this big (virtual) tree, we pairwise add up the shape functions as described above.

The open question for this approach concerns the type of the gray nodes in Figure 15, i.e. the type of the elements of our cell hierarchy. For the shape function arithmetic we only considered internal wiring and with that pink area only. On the other hand, we previously mentioned that the subcells in the cell hierarchy are red or eventually green during floorplanning (when we add the wiring channels to the subcell areas). With that, there may be three different hierarchical models depending on the color of the cell: a *hierarchical red model*, a *hierarchical pink model*, and a *hierarchical green model*. They will now be shortly discussed.

a) Hierarchical Red Model

Macro cells, i.e. layout boundaries, as well as the subcells which are used for floorplanning are red. They include all wiring to the cell boundary. It may be obvious that all elements of the

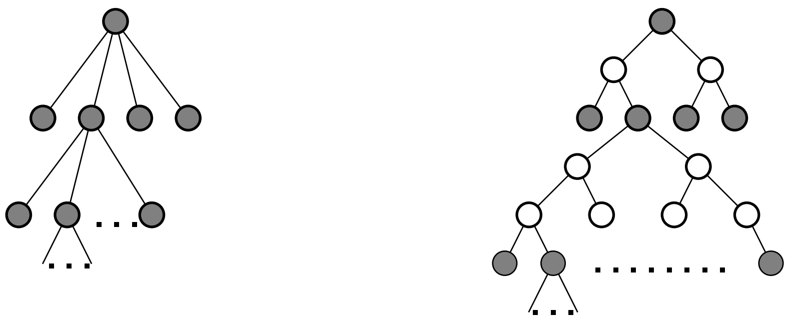

a) (n-ary) hierarchy tree *b) slicing tree*

Figure 15 *Cell hierarchy tree and corresponding slicing tree.*
During area estimation, we expand the n-ary hierarchy tree to a binary slicing tree. The shape functions of the gray nodes in the slicing tree are the shape functions of the corresponding nodes hierarchy tree.

cell hierarchy are modeled as red boxes. They can be defined recursively using equations (1), (2), and (3):

$$a^{red} = \sum_j a^{red}_{sub_j} + w^{pink} + w^{red} + a^{black} \tag{11}$$

This definition of a hierarchical model has two disadvantages. The first one is the fact that a net may pass through several levels of the hierarchy without connecting more than one internal pin, that means without pink segments (Figure 16). No good model is known for the estimation of a chain of red segments over several levels. The second disadvantage is that the area necessary for red wiring depends very much on the positions of the pins on the cell border which are not known during the bottom-up area estimation phase.

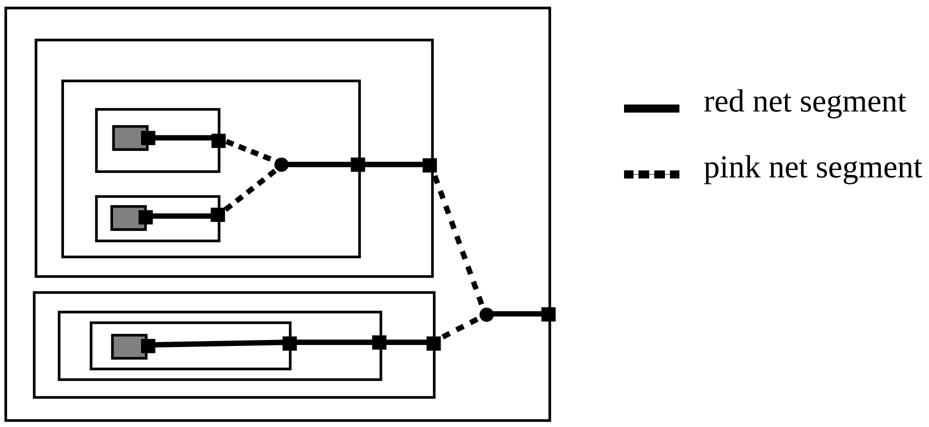


Figure 16 *Chain of red net segments*

b) Hierarchical Pink Model

Because of the disadvantages of the recursive red model, we use a recursive pink model during area estimation. In order to get rid of the red net segments, we add all red segments to the next pink segment on a higher level. Figure 17 explains this notion. The pink net segments and the pink boxes are shown by dotted lines. The extension of the pink segment yields an increase of the pink wiring area per net which is described by $\tilde{w}^{pink}$.

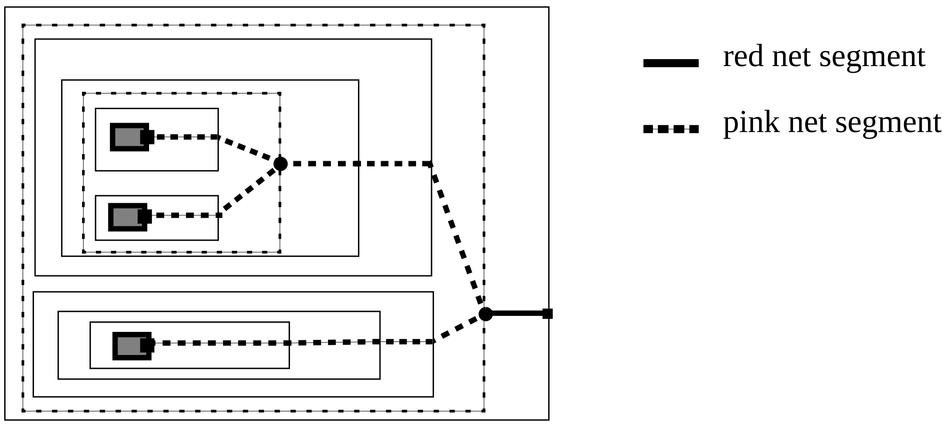


Figure 17 Pink net model.
This example has three pink net segments.

The pink recursion can then be described by the area equation

$$a^{pink} = \sum_j \tilde{a}^{pink}_{sub_j} + \tilde{w}^{pink} + a^{black}. \tag{12}$$

Our pink model has proven its quality for many standard cell and flexible cell examples. Some results are shown in Section 6. Our SHAPE FUNCTION GENERATOR therefore strictly uses the pink model for the complete slicing and hierarchy trees. In order to use a shape function for floorplanning at any hierarchy level, the additional red wiring space has to be added at that level:

$$a^{red} = a^{pink} + w^{red}. \tag{13}$$

Here, w^{red} is the red wiring area at the level of floorplanning only. We call this additional space the *red shift*. It will be discussed in the following Section.

c) Hierarchical Green Model

We mentioned above that for computing a correct geometry during floorplanning green boxes are needed. The green area a^{green} includes all subcell areas a^{blue} and the whole wiring area $w^{pink}+w^{red}+a^{empty}$ of the CUD. With that, it is easy to see that:

$$a^{green} = \sum_j a^{green}_{sub_j} \tag{14}$$

This is the most simple recursive equation. Nevertheless, the hierarchical green model is not applied in practice because floorplanning will always be done at one hierarchy level only.

4.2.5 Red Shift

The reduction to internal wiring results in a uniform, correct hierarchical area estimation model. Only for the top-level cell and during top-down chip planning we need an estimation of the wiring area to the cell/chip border. In this Section, we will show that the layout area of a given cell often depends very much on the position of the pins. During top-down chip planning, we use a more precise model for estimating the subcell areas which consist of internal wiring **and** external wiring to the subcell border.

We are now interested in knowing the influence of pin positions on the overall area. At least the chip planner must know this effect because the planner must reserve room for the subcells in a floorplan including all wiring area inside the subcells.

In our first area estimation approach, we estimated the external wiring area of cells with random pin positions [Zim88]. This led to a shift of the shape function to the top and right which was the same for all points on the curve. However, a large set of measurements has shown that this assumption is not true. The measurements showed that the influence of restricted pin positions is too large to use random pin distributions for the estimation.

In our new approach, we prefer a two-phase estimation concept. During bottom-up estimation before chip planning (phase I) we estimate the cell area including subcells and internal wiring only. External wiring will not be considered (see above). This area is very close to the smallest possible layouts of a cell for all aspect ratios. The additional areas which are needed for external wiring will later be estimated by the chip planner (phase II).

External wiring enlarges cells in x and y dimensions. We basically estimate the additional area with a function that is explained below and illustrated by the curves in Figure 18. The curves describe the total cell side length x^{tot}/y^{tot} with respect to the number of pins (or external nets) which we describe by the letter π. π_t, π_b is the number of pins at the top and bottom sides and π_l,π_r are the pins at the left and right sides, resp.

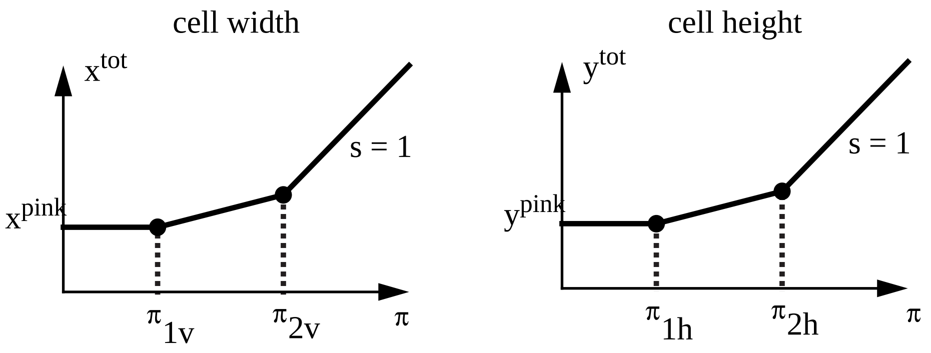


Figure 18 Cell dimensions with respect to the number of entering nets

In general, we have two similar curves for the horizontal and vertical sides of a cell. These curves consist of three segments which are separated by two points $\pi_{1v/h}$ and $\pi_{2v/h}$. The first segment is a horizontal line. Its value (x^{pink}/y^{pink}) is the width/height of the cell including internal wiring only. $\pi_{1v/h}$ represent the number of built-in feedthroughs which are not used for internal wiring w^{pink}. They can now be used for external wiring w^{red} without increasing the cell dimensions. As long as all built-in feedthroughs are not occupied the whole side length (x^{tot} and y^{tot}, resp.) is equal to the length of the side while considering internal wiring only ($x^{tot} = x^{pink}$ and $y^{tot} = y^{pink}$, resp.). For horizontal increase of the cell we consider vertical built-in feedthroughs *xf* only. The horizontal feedthroughs *yf* are used for computing the cell height.

The second curve segment represents an estimation method that is similar to estimating internal wiring. If we have more incoming nets than we can compensate by feedthroughs, we multiply each additional net by a special track demand factor t^{red} for external nets and, again, by the scale factor α representing the design rules.

For both segments together we estimate the total cell boundaries including external wiring as follows:

If $\pi_t \leq \pi_{2v}$ and $\pi_b \leq \pi_{2v}$ then:

$$x^{tot} = x^{pink} + \left(max(\left[(\pi_t + \pi_b) \cdot t_{xv}^{red} + (\pi_l + \pi_r) \cdot t_{xh}^{red} - xf \right], 0) \cdot \alpha_x \right)$$

If $\pi_l \leq \pi_{2h}$ and $\pi_r \leq \pi_{2h}$ then:

$$y^{tot} = y^{pink} + \left(max(\left[(\pi_t + \pi_b) \cdot t_{yv}^{red} + (\pi_l + \pi_r) \cdot t_{yh}^{red} - yf \right], 0) \cdot \alpha_y \right)$$

Let us look at the first formula in more detail. It computes the total cell width x^{tot}. As long as the number of vertical incoming nets $\pi_{t/b}$ is smaller than the second threshold value π_{2v} we add the pins at the top and bottom sides and multiply the sum with a horizontal track demand factor for vertical incoming nets ($(\pi_t + \pi_b) \cdot t_{xv}^{red}$). We similarly add the pins at the left and right sides and multiply the sum with a horizontal track demand factor for horizontal incoming nets ($(\pi_l + \pi_r) \cdot t_{xh}^{red}$)[3]. The sum of both terms is the number of necessary vertical tracks for external wiring which increases the cell width.

From that number we subtract the number of vertical feedthroughs *xf*. As long as these feedthroughs are not occupied, we can compensate these with external wiring. The cell width will not increase ($x^{tot} = x^{int}$). Since the number of built-in feedthroughs may be larger than the requirement for external wiring, we must take care that the subtraction does not become negative ($\rightarrow$ *max*-function.

Finally, we multiply the remaining track number with the scale factor α_x to get the additional width of the cell for the entering nets. The second formula computes the cell height y^{tot}. It is similar to the first function. We just use different track demand factors and the vertical scale factor α_y as well as the horizontal feedthroughs *yf* and the vertical side length y^{int}.

The third curve segment is needed for the case that the capacity of a side π_2 is smaller than the associated number of entering nets. The capacity of a side depends on the design style and must be determined experimentally (see below). If the number of pins on a side exceeds the capacity, each additional entering wire needs a whole track. The slope of the curve is therefore one. In the case that the number of entering nets at two opposite sides both exceed the capacity of the side (π_2) some entering nets can pairwise share a track. In this case we assume that the saturation point π_2 is not reached. We first compute the difference between the numbers of pins

3. Nets entering from the left and right sides also increase the cell width but less than vertically entering nets. At floorplan level we set $t_{xh}^{red} \approx 0,2$ and $t_{xv}^{red} \approx 0,5$.

at two opposite sides. Only for this difference we add whole tracks.

Design styles

As mentioned above, the track demand factors, the number of built-in feedthroughs, and the capacity depend on the design style. The also depend on the number of metal layers. Until now, we examined two different cell types and design styles. The first class are general cells for which we do chip planning. The second class are standard cell blocks.

If we have *general cells* and we allow over-the-cell routing, the functions for external wiring are the same as described above (figure 18). Before we have to extend the cell dimensions we fill-up the subcell feedthroughs which are not used by internal wiring. With two metal layers for routing, typically no built-in feedthroughs are left. The problem class is then the same as for chip planning without over-the-cell wiring. We can assume:

$$\pi_{1h} = \pi_{1v} = 0.$$

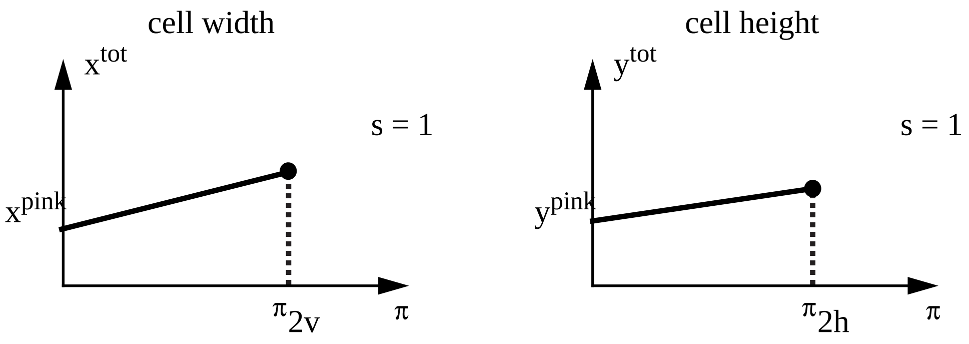


Figure 19 External wiring area for general cells without built-in feedthroughs The saturation point π_2 will never be reached for practical applications.

All entering nets need additional wiring space. For our test designs the entering net number π belongs to the second curve segment in all cases. The pin number will be multiplied with the track demand factors shown above.

Determining the saturation point π_2 is not so easy for general cells. We can only find π_2 if the general cell contains rigid macro subcells and the aspect ratio of the general cell is far away from one. In practice we never reached π_2. This has two reasons. First, pins occupy only a small amount of the frame which is below the capacity. The second, much more important reason results from the design style. As we mentioned above, estimating external wiring is important for top-down chip planning. In this case, the subcells of the considered general cell are not yet designed, their shapes are still flexible. They can easily be adjusted to the channel widths. In practice, we have so much flexibility for designing general cells that using the second curve segment, i.e. multiplying the number of entering nets with a track demand factor, is sufficient:

$$\pi_{2h}, \pi_{2v} \rightarrow \infty.$$

The second design style we examined experimentally are *standard cell block* layouts. We assume horizontal rows for this paper. Here it is reasonable to have no pins to the side of cell rows. An upper bound of the capacity of the vertical sides is therefore equal to y^{pink} minus the accumulated height of the rows. This is the sum of all channel heights.

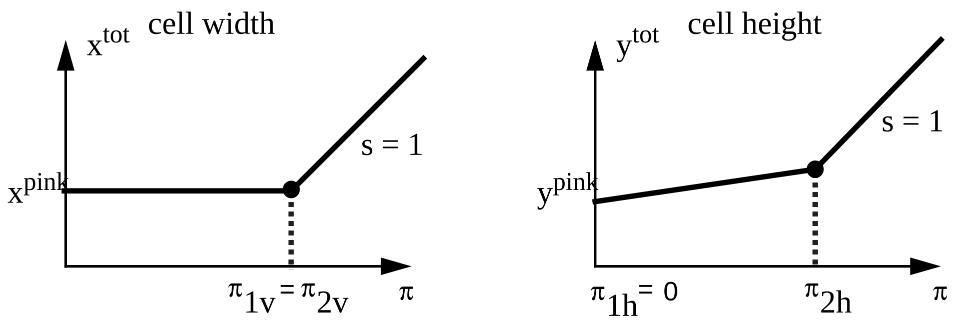


Figure 20 External wiring area for standard cell blocks

Experiments have shown that this value is too high for the threshold point π_2. We achieved good results by setting

$$\pi_{2h} = 0.2 \cdot (y^{pink} / \alpha_y).$$

Since standard cells do not have horizontal built-in feedthroughs, the function y^{pink} has the same shape as for general cells without built-in feedthroughs (figure 20). For the cell height we have

$$\pi_{1h} = 0.$$

For the cell width the conditions are different. Here we experimentally found the following: If we have a good standard cell placement tool, most of the standard cells with connections to the top and bottom sides of the block are not placed in the center of the block but near the top or bottom row. The occurrence of these cells decreases with their distance from the block border. On the other hand, built-in feedthroughs are used for internal wiring mainly in the center of the block.

The feedthroughs in the top and bottom rows are not needed for internal wiring. In the center where we have only few unused feedthrough we do not have many cells connected to external nets. In most cases, nets can enter from top and bottom without enlarging the block. Unused feedthroughs compensate the track requirement for external wiring.

Our experiments have shown that the number of feedthroughs at the outermost rows is an indication for the capacities of the top and bottom sides (π_{2v}). In the case that all these feedthroughs are used, each additional entering net needs a further feedthrough and therefore a whole track. On the other hand, as long as we still have unused feedthroughs, the width of the standard cell block does not increase due to external wiring. This means that we do not have the second curve segment for x^{tot} of standard cell blocks. π_{1v} is equal to π_{2v} in this case (Figure 20). For our standard cell placement tool and the currently used cell library we have

$$\pi_{1v} = \pi_{2v} = 0.5 \cdot (x^{int} / \alpha_x).$$

In our test designs, i.e. in practical applications, we very seldom reached this capacity.

4.3 Chip Planning

The purpose of chip planning is the placement of (rectangular) blocks, the determination of their shape, and global wiring. Global wiring determines channel or over-the-cell routing requirements, assigns nets to channels and determines pin-cost functions. The goals are minimal area, a solution to net length limits, and an estimation tolerance that allows chip assembly without iterations through chip planning and without loss in layout quality.

The major phases of the PLAYOUT CHIP PLANNER are *placement*, *sizing* and *global routing*. Especially for placement several algorithms can be selected. Placement can be improved by manual interaction. The CHIP PLANNER therefore has a menu-controlled user interface. The floorplan computation is supported by a number of important analysis tools. Besides the interactive control, the CHIP PLANNER can be controlled by a control command file. This is important for the design control by the design manager of the PLAYOUT CAD framework.

4.3.1 Placement

Placement should ideally not be separated from global routing. But it has been shown that already subtasks of placement are np-complete (see [Len90]). Therefore we even separate placement into several phases.

The restriction to slicing structures and its representation as slicing tree suggest as a first step the generation of the tree. The slicing geometry (topography) is fully described by the tree if it is oriented and ordered and if the leaf nodes are sized. In detail this means:

- Orientation: Every intermediate node is labeled v (vertical cut line) or h (horizontal cut line).
- Ordering: The ordering of siblings in the tree from left to right means ordering in the floorplan from left to right and bottom to top, respectively.
- Sizing: Dimensions (x, y) are chosen for all leaf cells. This is equivalent to choosing coordinates on the green shape function.

The topology is described by the oriented and ordered slicing tree. Without orientation and ordering we call it the unoriented tree. In PLAYOUT, the unoriented tree can be generated by clustering or partitioning. The CHIP PLANNER has a clustering algorithm as reported in [RRZ84] and a bipartitioning tool that implements the mincut algorithm [FiM82]. Cell area is used for balancing the tree. A disadvantage of theses simple methods is that the position of the CUD pins ($\rightarrow$ pin cost function) is not considered. To prevent this disadvantage, PLAYOUT further uses quadri and tri partitioning at the CUD level. For all possible topologies of four or three slices and for known CUD pin cost functions, the partitioning with pin propagation of the set of subcells is sought. Below this level bipartitioning takes place (considering pin location as much as it is possible without orientation). This generates a tree with an oriented ordered subtree at the root.

The topology is generated by either a combined orientation and ordering algorithm or by optimal orientation (Section 4.2.3, page 17) and a separate ordering algorithm. Both algorithms proceed in the same manner. In a breadth first traversal, subtrees with k levels are selected and in these subtrees all permutations are tested. Simple tests consider half-perimeter lengths of net bounding boxes. Better results have been achieved by calculating the shape functions for the roots of the subtrees for all permutations and then select the permutation with the area minimum. Runtime and quality increase with increasing k.

Placement is finished by sizing. Sizing calculates the shape function of the CUD for a given orientation and selects the CUD dimension closest to the given CUD frame. This sizing is simply the application of vertical or horizontal shape function adds in the topology. Note that the CHIP PLANNER has to add green subcell shape functions while the SHAPE FUNCTION GENERATOR calculates pink functions. The CHIP PLANNER also adds wiring space to correct for intersibling wiring and which results in pink CUD shape function. This will be corrected by a red and green increase for nets external to the CUD. This increase is distributed to all green subcells.

The resulting topography includes an area estimate for global wiring without exact knowledge of the wiring channel distribution. This will be refined after global wiring.

4.3.2 Global Wiring

Global wiring is performed on a channel intersection graph which is an abstraction of the channels to segments of the slicing lines. The channel widths are the capacities of the edges of the channel intersection graph. The nodes are the intersections of slicing lines. Figure 10 shows an example. For the purpose of over-the-cell routing, the channel intersection graph is extended by pseudo-channels. This can either be done by extending slicing lines across the CUD or by introducing edges across the sub-cells (dotted lines). Built-in feedthroughs are modeled by pseudo-channel capacities.

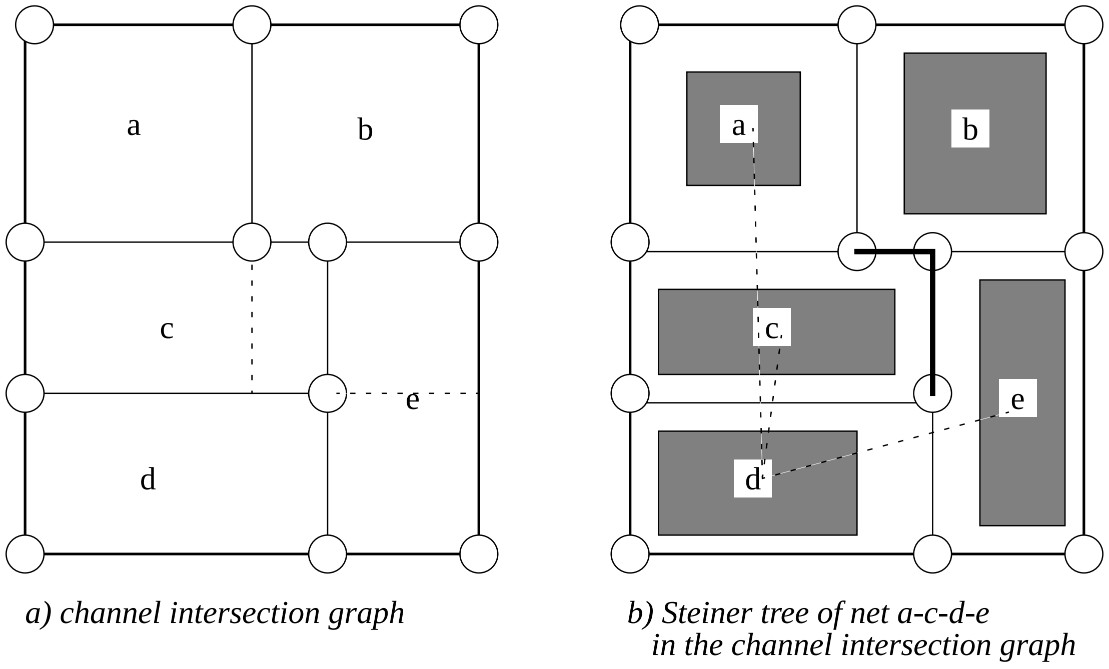

a
b
c
e
d
a) channel intersection graph
a
b
c
e
d
b) Steiner tree of net a-c-d-e
in the channel intersection graph

Figure 21 channel intersection graph to the floorplan from figure 10

Global routing does not find the best routing for a given channel capacity, but determines the channel capacities for the best routing path of each net. Only in the case of subcells with fixed over-the-cell routing capacities (e.g. macro cells), the router is constrained by channel capacities.

Global routing demonstrates one peculiarity of planning most clearly: decisions are deferred as long as possible if the information may still grow. Therefore, if a net has to be connected to a flexible cell, global routing terminates the net at a node of the channel intersection graph on the green outline of the cell. The pin assignment is deferred. Global routing uses a heuristic to determine a Steiner tree for each net on the channel intersection graph. At this point the green box is sufficient for routing. Exceptions are macro cells, because pin locations are fixed. The location of the red box of macro cells in the green box has to be estimated in order to find the corners adjacent to each of the pins. Global routing then proceeds as before.

Global routing results in net counts for each edge in the channel intersection graph (channel).Three types of nets have to be distinguished: A net that passes through a channel needs a full track. A net that enters a channel and ends there will need part of a track, determined by a track demand factor. A net which is local to one channel is called abutting and typically has a small track demand factor.

At first, all nets end at nodes of the graph and thus only passing nets occur. A net is abutting if it is degenerated to one node. It then has to be determined from the geometry of floorplan which edge has to provide wiring space. Determining the exact pin positions on the red boxes is deferred until the next lower level of the hierarchy is planned. Because these ending nets cannot be totally neglected, we add corresponding estimated area to the red box of the connected cell. After the numbers of tracks are determined, the width of all channels is calculated. This is followed by an optimal distribution of the channel areas to the green areas of the subcells.

Finally, pin intervals are determined for each pin that is not fixed. The pin interval is defined at the perimeter of the red box. If we move the pin out of this range, extra inter-subcell wiring will be necessary. This area has to be provided by the red cell. Reasons for moving pins out of the interval may be the pin capacity of the interval or the placement of the sub-subcells within the subcell of the CUD.

Figure 10 shows an example of pin intervals with four different cases. We assume two nets to be part of the floorplan of Figure 10. One net connects the cells *b*-*d*-*e*. The interval at cell *b* is the projection of the channel, at cell *e* the passing area, and at cell *d* the upper-right corner. The second net shows an abutment situation. Here, the intervals are the projection of the abutting cell sides on each other.

In conclusion, the results of global routing are:

- A more precise floorplan based on the additional knowledge of the channel widths.
- Pin-cost-function for all subcell pins.
- Assignment of nets to channels for detailed routing.

4.3.3 Global Wiring Iteration

Global wiring in the first iteration was based on a floorplan with estimated routing area requirements. The resulting floorplan may have different dimensions. In the general case the changes are not large enough to justify a new placement, but we can repeat global wiring for this new floorplan. This will again cause changes and may be repeated indefinitely.

These iterations can only have one goal: The improvement of the design. Currently we use

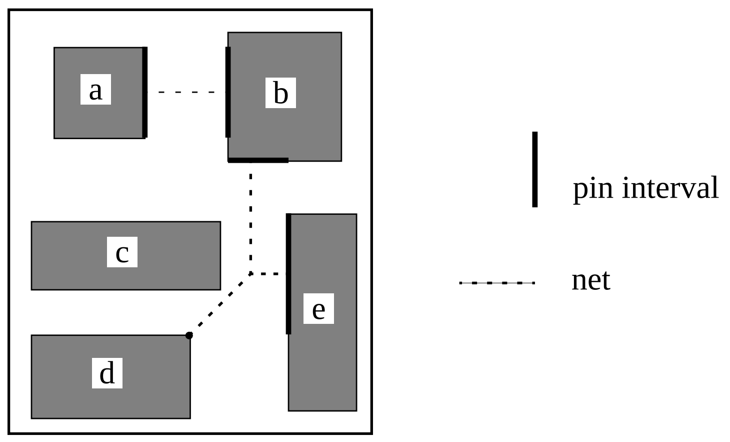


Figure 22 Pin intervals of two nets

the total CUD area as a measure of quality. Therefore we iterate as long as the area decreases. Our experience so far shows that a minimum is normally reached after one or two iterations. Also, the area gain caused by iterations is in most cases less than 5%. For global wiring iterations a wiring algorithm based on restricted channel width (from the previous iteration) may be reasonable. [Mei91] describes such an approach using flow methods.

4.3.4 Manual Interaction and Analysis

During the whole design process so far, the design decisions that affect the final layout the most is the placement phase during chip planning. Several combinations of placement algorithms and cost functions can be applied to find a good solution. No combination could so far be shown to be superior to others[4]. Thus one of the tasks of the designers is the selection of one or more combinations and the evaluation of the results. For this purpose a range of analysis tools is provided, e.g.

- Graphical display of the floorplan is used for visual inspection, determination of problem areas, and for interactive changes to the placement.
- Graphical display of the slicing tree with orientation and ordering. This shows problems in partitioning (unbalanced tree) and is used for more complex interactive placement changes as for example moves of subtrees.
- Area statistics showing the total area of selected nodes in the slicing tree and the shares of differently colored areas. This is used to roughly compare placement alternatives.
- Net lengths and a net density distribution histograms in the CUD rectangle.
- By selecting subcell or nodes in the tree, precise geometric dimensions and other details can be interrogated.
- Original and final shape functions can be viewed for selected nodes in the tree.

4. We did not experiment with tools like Gordian and TimberWolfMC because our goal is the development of the overall top-down design method not individual algorithms.

Currently, interactive changes of the placement are supported. The topography is automatically adjusted after every change and immediately displayed. In the future, we also want to support limited repartitioning steps if no floorplan with sufficiently small empty space can be found.

Interaction also allows moving back and forth between the different phases of chip planning. Also, intermediate states can be preserved and restored at any time. Thus the design can be interrupted and alternatives can easily be compared with the option to return to the best intermediate state.

4.4 3-Phase Chip Planning

The average tolerance of our area estimation method is about 10-15% [SuA95]. However, in the worst case the deviation can be substantially greater. In pure top-down chip planning, we have no chance to compensate these differences. The only possibility to react to deviations is to return to the shape function estimation phase and to change estimation parameters or to use realizations (layouts) of critical cells (using macros instead of flexible cells).

On the other hand, if we are not planning the top-most cell, it may be possible to compensate the deviation of the current cell shape with the deviations of the sibling cells. The area of the supercell should not change if the sum of all area estimations of the sibling cells is similar to the areas of the floorplans and layouts, respectively. Furthermore, all planned cells are slightly flexible because their subcells are still flexible. This flexibility increases the chance of a good balancing.

So, it is useful after computing a floorplan of the CUD to perform an *adjustment planning step* for the supercell before continuing the top-down planning process at the subcell level. The deviations at the current level can be compensated at the supercell level. Figure 23 depicts that approach. There are three planning steps which we denote by α, β, and γ. After planning the subcells at level i+1 (phase α), we perform an adjustment step at level i (phase γ) before con-

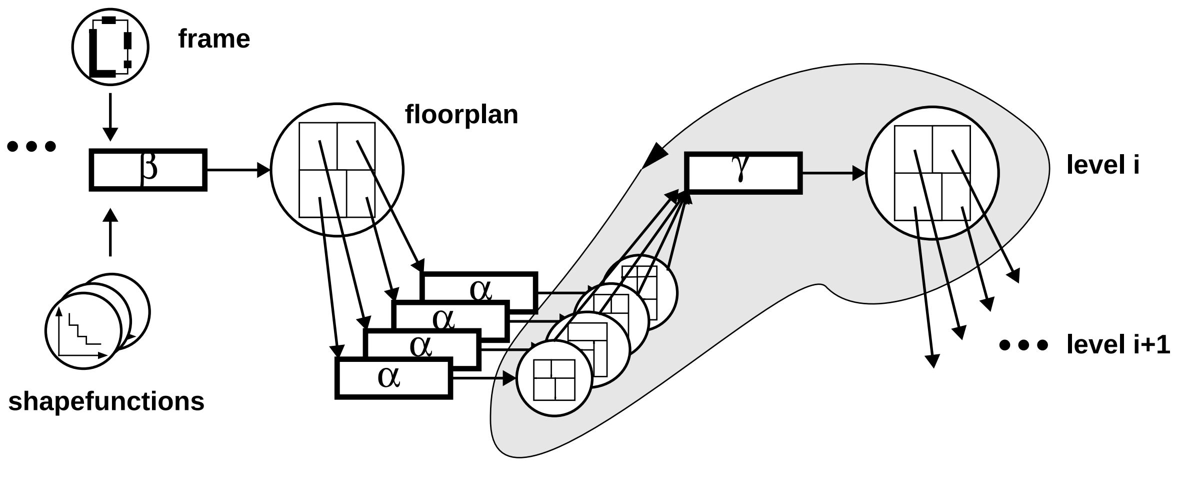


Figure 23 Floorplan djustment process

tinuing the top-down planning of all subcells at level i+1 (phase β).

Of course, the cell at level i itself is part of an adjustment process for it's supercell on level i-1. Thus, the three fundamental planning phases α, β, and γ are performed with each cell. Therefore, we call our planning method *Three-Phase Planning* [SAZ92].

The following actions are performed:

Phase α *(Initial Floorplan):*

- placement
- global wiring
- wiring area estimation
- sizing (computing a correct geometry and a shape function for the CUD)
- computation of pin constraints for the CUD

The result of phase α is a floorplan from which we use the topology for further planning steps and the pin constraints of the CUD for an adjustment step at the supercell level. Since the subcells are still flexible, many floorplans with different shapes but the same topology are feasible. We generate a *refined shape function* that describes all possible shapes of the same topology. This shape function is a very important input to the supercell adjustment (phase γ), too. Refined shape functions are more precise than shape functions of flexible cells because they rely on a particular topology and a global routing (not only on a rough wiring area estimation).

Phase β *(Adjustment to new Frame):*

- global wiring (using the placement of step α and the frame description from the supercell adjustment step γ)
- wiring area estimation
- sizing (computing the subcell shapes)
- computation of pin constraints for the flexible subcells

As we will see later, in the adjustment step γ of the supercell, a new frame of the CUD was computed based on its refined shape function and its pin assignments of phase α. In phase β, we adjust the geometry of the topology from phase α to this new frame.

As in the pure top-down planning strategy, we now compute the pin constraints for the subcells. The subcells can then be planned in phase α (see above) which results in cells with new constraints (refined shape functions and pin constraints). These constraints are input to the CUD adjustment planning phase γ.

Phase γ *(Adjustment to more precise Subcell Data):*

- correction of the global wiring (because subcell pins can move)
- wiring area estimation
- sizing (selection of subcell shapes using the refined shape functions of the subcells

from phase α)

- computation of pin constraints for the subcells

The resulting subcell constraints (area, shape, and pin positions) of phase γ are used as input to phase β and γ of the subcell.

In addition, our Chip Planner allows the designer to interactively change the floorplan in every planning phase. For example, it is sometimes necessary to change the slicing structure (and with that the topology data) without changing the adjacencies of the cells, i.e. the global topology. The global wiring may use new channels for a particular net when the topology of the slicing lines changes.

In figure 24 the three planning phases with the top-down and bottom-up data interchange are outlined. In contrast to our refined shape functions all other hierarchical top-down design sys-

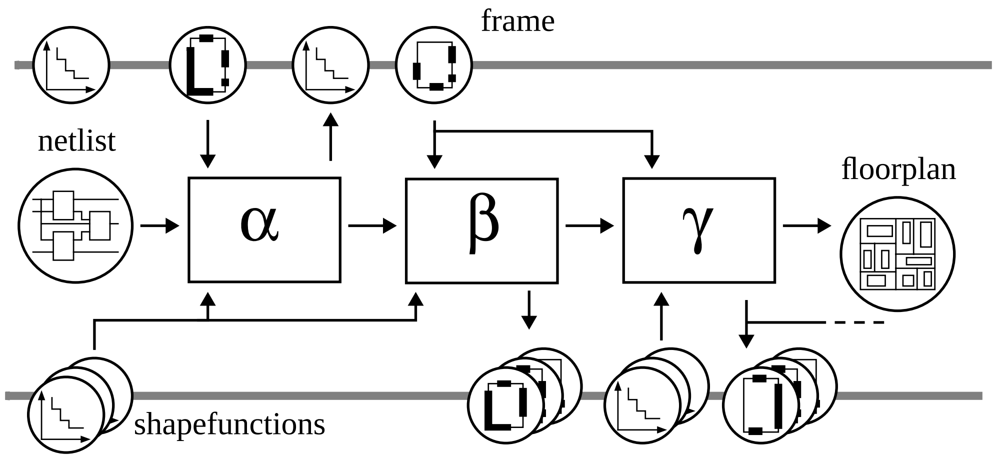


Figure 24 Chip Planner input/output during the three phases (a floorplan is also passed from phase α to β and from β to γ)

tems use at most one fixed shape for an adjustment [YiW89]. They do not provide any top-down adjustment like the phase β.

Figure 25 shows an abstract notation of a hierarchical planning over three levels. The plannings of all subcells were combined in each case. Even though there are bottom-up movements, it is obvious that the global direction of the design process is still top-down. At the top level, the pad frame is created (e.g. by our graphical Pad Frame Editor; *pfe*) and at the lowest hierarchy level we perform the cell synthesis *syn*.

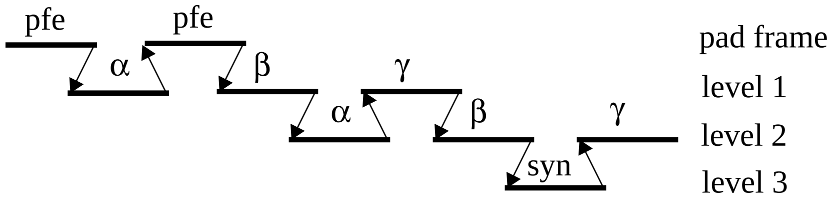


Figure 25 Three level chip planning with adjustment

Stepwise Refinement

So far we assumed that all planning steps α of the subcells will be performed in parallel, independently of each other. However, in several cases, balancing the floorplan in phase γ can be improved by generating the floorplans of the subcells step by step. On the other hand, it may be desirable to insert all critical cells first (e.g. cells which do not fit to the estimations) and to leave the remaining cells flexible which can balance the whole floorplan in a following adjustment step.

An extreme possibility for a stepwise refinement strategy is the depth-first traversal of the hierarchy tree. The advantage of this method is that we have the freedom to build critical modules (subtrees) step by step first. The disadvantage of a depth-first strategy is that we loose the whole parallelism. No planning processes can be performed concurrently.

Another possible strategy is to plan all subcells (perhaps with different topologies for each subcell) and leave the final configuration decision to the CHIP PLANNER. Within the adjustment phase γ of the CUD, the CHIP PLANNER chooses the most fitting floorplan alternative of each cell. Experiments have shown that there is no single best solution. The strategy with best results depends on the actual design.

Convergence

In total, there are many possible strategies although we use only three different phases. In an extreme strategy, it is possible to descend and to ascend the hierarchy tree in a yo-yo fashion.

So, we have to show that our procedure terminates. In each bottom-up adjustment step γ, we always replace at least one flexible subcell by a floorplan or a layout. A floorplan will be replaced by a layout only. When all flexible subcells are replaced by a layout, the final cell assembly terminates the adjustment procedure.

While the procedure terminates for all strategies, we can see a convergence behavior when executing the stepwise refinement strategy. Here, we replace the inexact flexible cells with more exact rigid cells step by step. With each replacement, one inexact component has been removed and with that the tolerances will become smaller. The floorplan geometries converge to the final layout.

Restrictions

In spite of all the freedoms we have, we are not permitted to build a cell and one of its subcells in parallel. Before we can make a planning step γ_k, we have to stop all subcell planning processes and have to collect the current results as input for the adjustment step γ_k. After γ_k, we can go on with the subcell plannings in consideration of the new constraints from γ_k. If we do not stop the subcell processes, we build divergent bottom-up and top-down frame descriptions.

It is the design managers task to keep track of the planning process. In [ASZ93] we describe seven rules which are implemented in our design manager. These rules guarantee the correct traversal through the circuit hierarchy. They are the basis of an automatic design management that controls a correct design flow.

Three of the seven rules correspond to the three chip planning phases.

- Rule 1 describes that phase α of a CUD can be performed as soon as a frame has been computed top-down (by a planning step of the supercell) and no floorplan of the CUD has already been produced.
- Rule 2 represents phase β in a similar way. The only additional precondition is that phase α has already been performed. After computing the floorplan of phase β, the frames of the subcells are available for which we can now perform phase α.
- The precondition of rule 3 is that all subcells are ready for an adjustment step. This means that we already have a refined shape function for each of the subcells available. Executing this rule enables rule 1 or rule 2 for the subcells.

Using these three rules, an adjustment in phase γ is only possible when all subcells have been planned in phase α. On the other hand, it may be desirable to perform phase γ before all subcells are planned or to use a subcell floorplan of phase γ for the adjustment (see: stepwise refinement strategy). In this case, we have to ensure that these subcells are not within an active planning process. All active design processes of the subcells must be stopped before phase γ of the CUD can be performed. We need four further rules to stop the subcell processes.

- Rule 4 starts stopping for all active subcells processes.
- Rules 5 and 6 recursively stop all active processes in a whole circuit subtree. The recursion is terminated by applying rule 4.
- Rule 7 signals that all subprocesses are terminated and (re-) activates the adjustment phase γ.

4.5 Chip Assembly

Chip assembly is the final phase of layout synthesis. Chip assembly in PLAYOUT is especially designed for our top-down chip planning design style. Three-phase chip planning and chip assembly have a close interaction to guarantee an exchange of constraints between levels of the hierarchy. During chip assembly the floorplan is refined and the layout completed. Chip assembly is composed of two different functions: *cell synthesis* and *cell assembly*. Different strategies can be applied for moving in the hierarchy tree. Instead of generating the layouts of subcells in the same floorplan independently in all cases, layout may also proceeds in parallel and constraints like pin positions or shape and position of the blocks in the floorplan may be exchanged dynamically. This method results in excellent adjustment of pin positions between cells and thus a reduction of channel widths. The compaction problem as part of chip assembly is solved by a genetic algorithm.

Besides our approach, there is only very little related published work about chip assembly in conjunction with top-down chip planning. Two other systems should be mentioned. In the BEAR system [PDM90] floorplanning and chip assembly are integrated. The floorplan is refined until all channels can be routed. The CHEOPS system [MBE90] has manual floorplanning but an explicit chip assembly step.

4.5.1 Strategies

There are three possible strategies for chip assembly: *pure bottom-up*, *iterative*, and *parallel*. *Pure bottom-up chip assembly* means that the layout of all cells at any level of the hierarchy are completed independently of each other, either by cell synthesis or by cell assembly. Thus the layouts can be executed in any order or concurrently.

Iterative chip assembly is a continuation of our iterative three-phase chip planning approach. It is identical to the adjustment phase γ while using subcell layouts instead of subcell floorplans for the adjustment. But the main direction is reversed. We now start at the lowest level of the hierarchy. For one or more selected cells the layout is finished according to the current floorplan. Then the floorplan is adjusted by executing the sizing step again and correcting the global routing. This may change the specifications for the remaining cells, that are still flexible. As we stated above, the quality of the layout depends very much on the order of the subcell refinement. Our experiments show that critical cells should be laid out first because less critical cells can better be adjusted in general case. To date, there is little more knowledge about a good order.

Parallel chip assembly tries to avoid the order dependency. Constraints are passed between cells while layout is performed. Constraints are shapes, pin intervals, and exact locations of the cells relative to each other. As layout of a cell proceeds, all three groups of parameters change from planned to fixed. The occurring changes in these parameters may change the requirements of other cells. If, for example, two cells share a channel, then pins of a net on opposite sides of the channel should abut. This can be achieved by initially letting the pin positions float and attract each other during parallel layout. Until now, we manage parallel cell synthesis. Parallel cell assembly seems possible but is not yet implemented.

4.5.2 Cell Synthesis

Cell synthesis is generating the layout of a CUD with primitive subcells. Examples of subcells are standard cells, sea-of-gates cells, or transistors in a gate matrix design style. Tools can be placement and routing tools or module generators. In any case the CUD has been represented to chip planning by a shape function. The CHIP PLANNER selected a point from that curve, assigned pins to pin intervals, and adjusted the shape according to the number of entering nets ($\rightarrow$ red shift).

In PLAYOUT only standard cell placement and routing is currently implemented. The implementation of the placement follows the simulated annealing algorithm as implemented in TIMBERWOLFSC [SeL87]. Improvements have been introduced in row length adjustment by dynamic estimations of feedthroughs. The width of the channels is continuously estimated to improve the net cost function. This is all done by a fast global routing step which can be applied during placement.

Influence of pin intervals

The influence of pin intervals on the CUD frame is an important issue for our top-down approach and needs to be explained in more detail. Pin intervals are determined by the CHIP PLANNER after global routing. A pin interval is the part of the cell boundary, where the cost for connecting an external net to the cell boundary is a zero (see Section 4.3.2).

Let us consider cell *e* from of 10 as an example and let us assume that the cell internally may want to place the pin outside of the interval. This is possible but results in a penalty for the internal net costs. For cell *d* it is impossible to place a net directly in the corner. To achieve pin positions close to the corner the same penalty applies. This penalty is calculated during standard cell placement as the smallest Manhattan distance from the rectangle that encloses the inner terminals of a net and the pin interval. In Figure 26 this distance is shown as a double arrow.

Final pin placement is done before global routing. For a corner interval a pin position is selected which is closest to an inner terminals of the net but belongs to a side adjacent to the corner. For other intervals we must distinguish if the enclosing rectangle of the inner net terminals matches the interval or not. In the first case the best position inside the interval is selected, thus following the global routing constraints on the next higher hierarchy level. If it does not match we have the choice to select the best position on the side of the interval (arrows *a* and *b* in Figure 26) or to select the best position inside the interval, depending on available routing tracks (arrow *c*). In figure 26 three possible placements for a pin are indicated by the dotted arrows. If the pin is placed inside the interval, an additional vertical routing track (feedthrough) must be available on the fourth row.

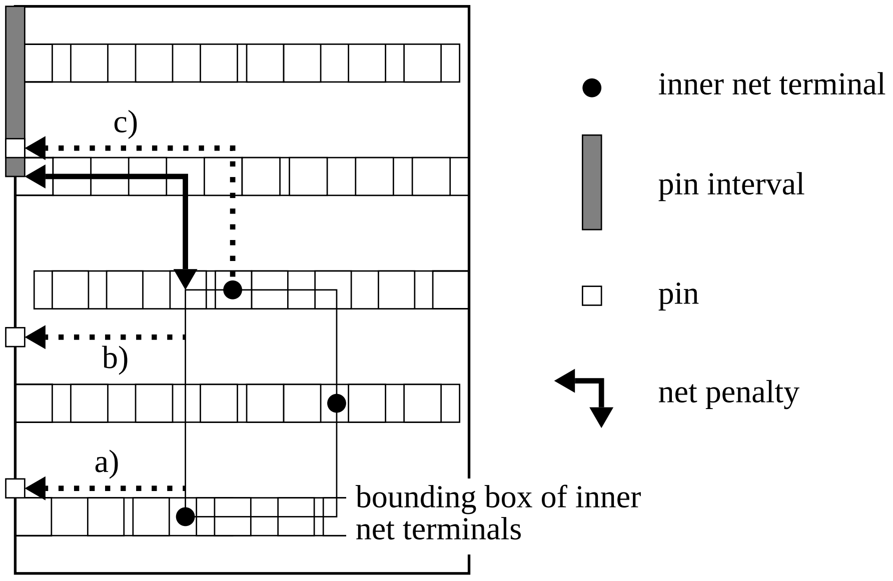


Figure 26   Net penalty due to pin interval

Parallel cell synthesis

One goal of parallel cell synthesis is the reduction of the width of channels between cells by matching pin positions on opposite sides of the channel. The width is only effected by nets that have all pins in the channel (→ abutment nets). The net between cells *a* and *b* in Figure 10 shows such a situation. The global router assigns the overlap of the cells as pin intervals. If the layouts of the cells would be generated independently of each other, any position of the pins might result.

A second goal is the fine-tuning of shapes and the floorplan positions of the cells. As the positions of the standard cells converge to their final values, the sizes of all cells become more precise and rigid. Also, the width of the channels can be calculated more precisely as pin intervals shrink to dots. This requires a continuous adaption of the floorplan.

In `PLAYOUT` we built a multi-process environment which allows constraints to be propagated between processes at the same level of the hierarchy and with a parent process at the next higher level. All adjacent cells exchange pin intervals, which influence the penalty added to the internal net cost. The result of the iteration process is that the enclosing rectangles in adjacent cells attract each other and hopefully overlap in the end. Global routing will then select pin positions in the overlap so that abutment is guaranteed or as close to each other as possible. The width of the channel between the cells is minimized. But it has to be mentioned that the additional constraint on the pin intervals may result in larger cells. Experiments so far show that the overall gain is larger than loss.

It is the task of the parent process to control the overall cell placement. This is done by adapting the floorplan to the intermediate results of the cell synthesis processes ($\rightarrow$ placement). Because the actual shapes of the cells generally diverge from the predicted shapes, it is necessary to legalize the channel geometry of the floorplan. This results in new subcell positions which must be passed to the synthesis processes. The operation has to be performed repeatedly.

In the case that actual and predected shapes of cells diverge drastically, it may also be necessary to resize the floorplan completely (while keeping the given topology) thus getting new shapes for the subcells. The problem of propagating these new shapes to the child processes is that as soon as the number of rows of a standard cell block is fixed, the shape has little flexibility. If the parent process forces a drastic change in shape, the number of rows has to be changed and the standard cell placement process has to be started again. All other processes which are not affected will be stopped until this cell is in the same state as before.

Parallel cell synthesis is an excellent candidate of a distributed computing system. All processes are highly CPU-intensive and constraints propagation requires relatively low communication bandwidth. Therefore a local area network and the UNIX Network Computing System (NCS) are used as communication environment.

4.5.3 Cell Assembly

Given a floorplan and set of macro cells as subcells, cell assembly is the task to adjust the placement of the subcells in a way that detailed routing can be completed with minimal routing and wasted area. After layout synthesis of the subcells has been completed, the floorplan is no longer optimal because the sizes and the shapes of cells generally have changed. A final adjustment step of the floorplan is necessary. For that we have the choice to simply legalize the channel geometry or to do a further optimization step.

The legalization is done as follows. The width of the channels can be calculated more precisely than after global routing because the exact pin positions of the subcells are known. We first calculate the density for all channel segments between adjacent pairs of cells as a function of their relative position. For each pair of cells we get a density function as shown in figure 27 [Bec93]. Legalization is done by adjusting the channel topography bottom-up the slicing tree. Channels are equivalent to slicing lines and are extended or reduced until all distances between adjacent cells at least are equal to the density function for the actual positions of the cells. Since the subslices have already been fixed, individual cells along a slice cannot be moved relative to the others.

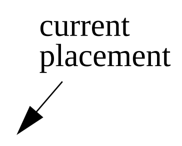


Figure 27   Channel density function

The second choice is an optimization step that moves cells together as close as possible. Since the channel density function is not a constant, the optimization can determine the best position of cells along a channel which need not be the position after legalization. This optimization problem is not just two-dimensional compaction (which is difficult enough) but the distance between cells is also a function of their relative position perpendicular to the distance. The problem is reduced because the topology must not change. The range of expected shifts between cells is limited by the distance functions. Our solution to the problem is a genetic algorithm. A population of approximately 100 legal floorplans is used to generate children by crossing and permutation. A selection process chooses a new generation of floorplans from parents and children including the best floorplans of the parent generation. This is repeated until the solution does not further improve [GHZ91].

Channel definition and pin positioning

The final task before detailed routing is the geometric definition of the channels and the assignment of junction pins at the end of the channels. Since we use slicing structures we need not have switchboxes as shown in Figure 10 but we define channels along the whole slicing lines.

A still open problem is the definition the geometrical boundaries of the channels because the slicing lines are only of topological interest. We have to define where the channels precisely intersect. We decided to define the channels *as small as possible* to increase the optimization potential of the channels of ancester nodes in the slicing tree. Figure 28 shows an example of our approach. The bold-faced polygon encloses the routing area corresponding to the horizontal slicing line. The segments of the polygon which do not belong to the border of a cell intersect the routing area from orthogonal channels. On these segments virtual pins are defined which are called junction pins. All these junction pins are located to fixed positions which results in a switchbox with irregular shape and connections on all segments of the polygon.

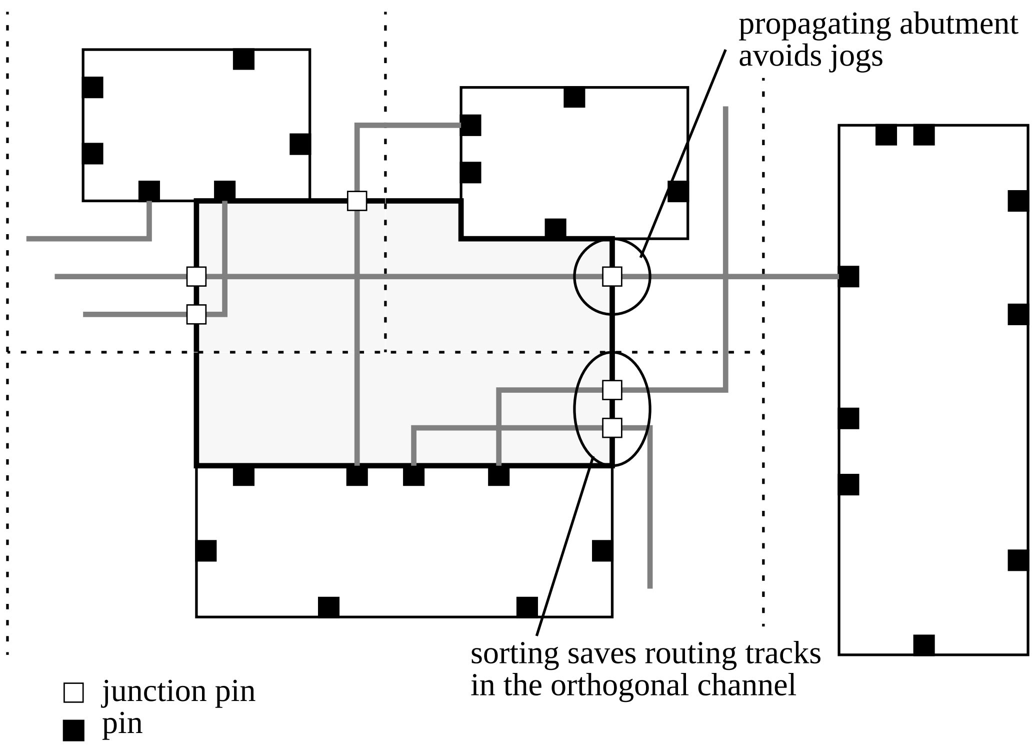


Figure 28   Channel definition

The placement of the junction pins is performed by sorting the nets and by propagating abutment positions and some other constraints. We implemented the net ordering algorithm from [Gro89] to avoid unnecessary crossings but did some slight modifications. Abutment checks which lead to fixed positions as indicated in Figure 28 will be preferred in case of a conflict with sorting. Our main goals are:

- minimizing the number of jogs in a net
- well structured channels which support easy routing in Manhattan style routing model
- minimizing the overall net length
- routing of all channels in parallel.

A disadvantage of using such an ordering algorithm is the requirement for a switchbox router. However, since our channel definition needs a switchbox router anyway, we can make full use of the junction pin assignment method. Experiments with the assignment method have shown an astonishing reduction in wire length, number of vias, and routing effort.

5. The PLAYOUT CAD Framework PLAYFRAME

The PLAYOUT design system has been extended by a CAD framework when it became necessary to conduct large test designs with several hierarchy levels in a university environment. Since our group always has limited resources of CAD tool developers and designers we decided to increase the CAD support for the development of the PLAYOUT system itself as well as for the test designs as much as possible. This was the beginning of our CAD framework project. The framework is called PLAYFRAME. The project contains research in the fields of databases, design flow management, design planning, and tool integration.

The PLAYFRAME architecture is very similar to the general framework architecture which has been described by Newton et. al. in [BHN92] (Figure 29). The kernel of PLAYFRAME is an object-oriented, prototype design database which has been developed in our group. However, our main research topic in data management was on data modeling. Our first data model which was published in [SiZ89] was extensively tested by many test designs. Recently, we improved the model especially for a better support of top-down design steps [SAS94].

Besides database support, the administration and control of the design flow became more and more important with increasing circuit sizes. Our large test designs have shown that it is almost impossible to conduct such designs without computer support. PLAYFRAME has therefore been extended by an interactive design manager called DESIMA.

The third major topic of our framework research is a generator-based support of the CAD tool development. A new software engineering environment called MOOSE was first developed to automatically generate the data management components of our CAD tools from abstract models. Due to the success of this approach we now extend MOOSE to generate other software components (e.g. the tool flow control and the graphical interface), too.

These are the three major research topics of our CAD framework project. They will now be described in some more detail.

5.1 The Design Database PLAYBASE

Since the main focus of our PLAYOUT project is the design of very large, hierarchical circuits, it very early became necessary to have access to database support for managing the large

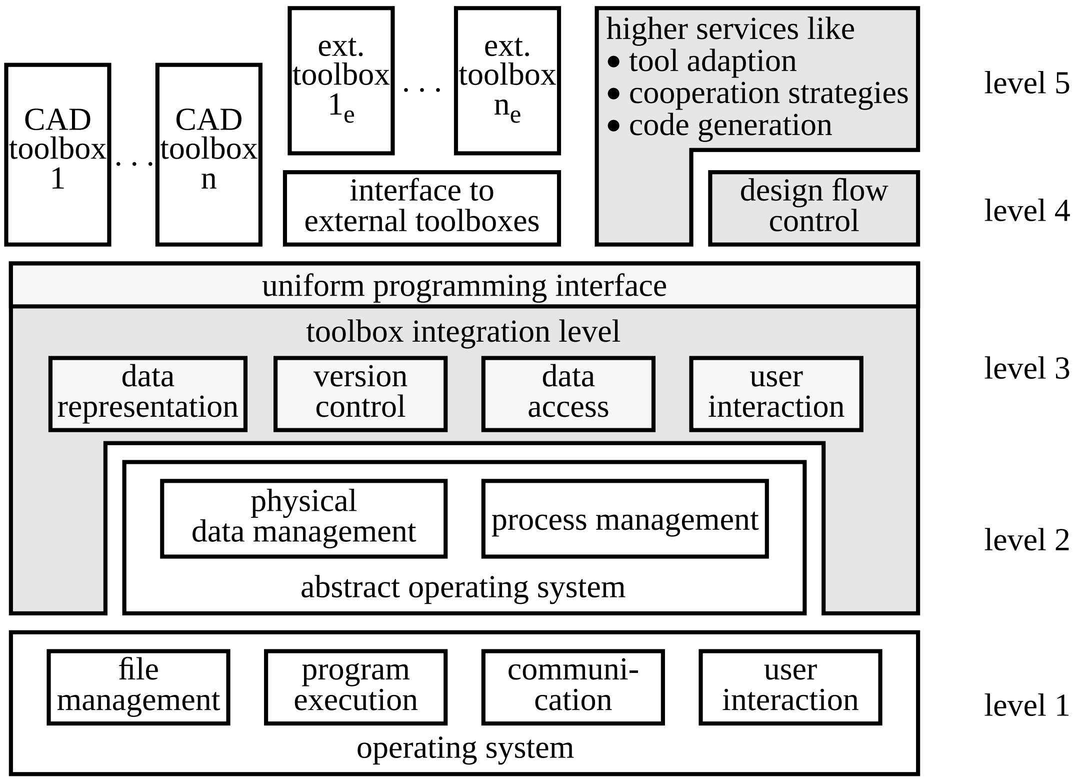
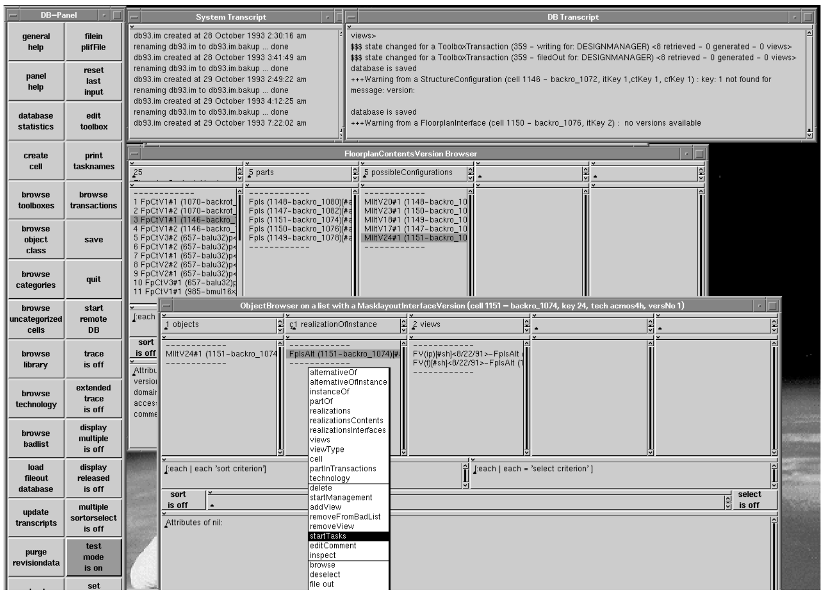


Figure 29 PLAYFRAME layer architecture

amount of design data. This requirement was already existent in the mid 1980's when no commercial non-standard database was available on the market. We therefore had to design and to implement our own prototype design database. Our major requirement was an easy to use data administration system with a comfortable interface. Less important were the permanent data security and the multi-user ability of the database which are two important aspects for commercial databases. We do not need these aspects because none of our test circuits has to be designed as in an industrial environment. Our main focus therefore was the data modeling and the administration of long-term transactions which are necessary for design as well as a comfortable access to the data basis.

We implemented our prototype database PLAYBASE in SMALLTALK-80, an object-oriented programming language. The advantage of this approach was the comfortable programming environment of SMALLTALK-80 which was a real help for the prototype implementation of our object-oriented data model [SiZ89], the transaction management [Sie89], and an interactive user interface based on a universal data browser (Figure 30).

The SMALLTALK-80 database prototype is very successful as long as all data can be stored in main memory. In the case that the SMALLTALK-80 image does not fit into main memory the performance becomes unacceptable. To ensure that the large amount of design data which are necessary for our designs can be managed by our database implementation, we partitioned the design data in *meta data* and *micro data*. The objects and the relations in the database only concern the meta data (administration data) which are most important for the design manage-



```
db93.im created at 28 October 1993 2:30:16 am
renaming db93.im to db93.im.bakup ... done
db93.im created at 28 October 1993 3:41:49 am
renaming db93.im to db93.im.bakup ... done
db93.im created at 29 October 1993 2:49:22 am
renaming db93.im to db93.im.bakup ... done
db93.im created at 29 October 1993 4:12:25 am
renaming db93.im to db93.im.bakup ... done
db93.im created at 29 October 1993 7:22:02 am
```

```
views>
$$$ state changed for a ToolboxTransaction (359 – writing for: DESIGNMANAGER) <8 retrieved – 0 generated – 0 views>
$$$ state changed for a ToolboxTransaction (359 – filedOut for: DESIGNMANAGER) <8 retrieved – 0 generated – 0 views>
database is saved
+++Warning from a StructureConfiguration (cell 1146 – backro_1072, itKey 1,ctKey 1, cfKey 1) : key: 1 not found for
message: version:

database is saved
+++Warning from a FloorplanInterface (cell 1150 – backro_1076, itKey 2) :  no versions available
```

```
25
------------
1 FpCtV1#1 (1070-backrot_
2 FpCtV1#2 (1070-backrot_
3 FpCtV1#1 (1146-backro_
4 FpCtV1#2 (1146-backro_
5 FpCtV3#2 (657-balu32)p
6 FpCtV1#2 (657-balu32)p
7 FpCtV1#1 (657-balu32)p
8 FpCtV2#2 (657-balu32)p
9 FpCtV2#1 (657-balu32)p
10 FpCtV3#1 (657-balu32)p
11 FpCtV1#1 (985-bmul16x

5 parts
------------
Fpls (1148-backro_1080)[#a
Fpls (1147-backro_1082)[#a
Fpls (1151-backro_1074)[#a
Fpls (1150-backro_1076)[#a
Fpls (1149-backro_1078)[#a
------------

5 possibleConfigurations
------------
MlltV20#1 (1148-backro_10
MlltV23#1 (1150-backro_10
MlltV18#1 (1149-backro_10
MlltV17#1 (1147-backro_10
MlltV24#1 (1151-backro_10
------------
```

```
1 objects
------------
MlltV24#1 (1151-backro_1074
------------

c1 realizationOfInstance
------------
FplsAlt (1151-backro_1074)[#
------------

2 views
------------
FV(ip)[#sh]<8/22/91>-FplsAlt
FV(f)[#sh]<8/22/91>-FplsAlt (1
------------

[:each | each 'sort criterion']
[:each | each = 'select criterion' ]
Attributes of nil:
```

Figure 30   PLAYBASE graphical user interface

ment. The micro data which are the basic design data like nets, pins, etc. are stored in regular files in the file system of the operating system. These files will be accessed by the database only. For preparing the input of a design step, PLAYBASE combines and extends these files by special header informations with respect to the meta data model. This approach can be seen for most other CAD framework approaches, too. With progress in object-oriented data base research it may be possible to store both, the meta data and the micro data, together in the database.

The transaction model that is implemented by PLAYBASE is a subset of the transaction models of most advanced design databases. We also implemented long-term transactions which do not follow the ACID principle anymore (see Figure 31). Design transaction may last hours or days and may manage large amount of complex data.

In our prototype database we did not implement all aspects of the right column in Figure 31. On the one hand, we do not allow nested transactions and on the other hand, our design data are stored persistently only after savepointing which has to be done explicitly by the user. All further security aspects are those of the operating system. However, these restrictions do not limit our test designs. Until now, there was no need for nested transactions. All further splittings of the transactions could easily be done outside the database. The persistency of design data based on savepointing is sufficient for our environment. It is even questionable if a more complex transaction model would be more helpful.

The most positive feedback from our designers concerns the interface of PLAYBASE. On the one hand, the extended SMALLTALK-80 browsers shown in Figure 30 are very suitable for the list-oriented, interactive navigation through the design data. On the other hand, the flexible programming interface can be easily exploited for implementing a prototype design flow manager directly on top of the database [SpS88].

ACID feature	standard transactions	design transactions
atomicity (A)	- atomic => either all changes are visible (→ commit) or none is visible (→ abort)	- saving as much of the work as possible - intermediate states are possible → save points
consistency (C)	- consistency checks are short, simple, and static (e.g. fee > 0)	- consistency checks are expensive (e.g. checking if a layout implements a given function needs extraction and simulation tools)
isolation (I)	- intermediate states are invisible to other transactions	- nested transactions possible
durability (D)	- committed changes survive system crash	- committed changes survive forever → version concept
amount of data	- few records	- large amount of complex data → check-out/check-in
deadlocks	- abort of transaction	- resolved by designers

Figure 31 Transaction features

5.2 Design Flow Management and Design Planning

Currently, we have a second version of a design manager as part of our framework. While the first version was implemented together with the database in the same SMALLTALK-80 image (which realized tightly coupled systems), we now have a separate design manager which interacts with PLAYFRAME via a remote procedure call (RPC) interface. This interface is transparent to the designer who still believes to interact with the database directly (but more comfortably).

One of the most important aspects of a good design management tools is a suitable visualization of the actual design state. For evaluating the design state the list-oriented browsers of the database are not sufficient. Important design objects must be graphically visualized in their typical representation. Additional analysis tools are necessary to compare design alternatives. In addition, the graphical interface must be able to support the designer to interfere in the design in an easy manner. This emphasizes the importance of the user interface of the CAD framework. Furthermore, additional planning and control assistants are necessary to guarantee a consistent and successful design. The final decision will still be made by the designer.

For visualization of the design state we implemented a design management component called DESIMA [Que94]. It has a graphical interface for representing design objects. As stated above, a textual representation of the design objects and their relations, which is typical for databases, is not sufficient for estimating the design state. All design objects should be represented in a suitable, graphical manner (Figure 32). DESIMA, for example, represents the hierarchy relations by trees (as it is also done by the Repartitioner); circuit data, floorplan data, and layout data are shown as schematics, floorplans, and layouts while the visualization abstracts from unnecessary details.

Figure 32   DESIMA user interface

The graphical representation of all design objects which are important for the design state enables the designer to comfortably compare design alternatives. `DESIMA` further supports this comparison by integrated analysis tools. Very helpful or even necessary is the fact that `DESIMA` manages the design data consistently in several windows in an object-oriented fashion. Version and domain relations will be considered. For instance, if the designer selects a node in the hierarchy tree, then `DESIMA` also highlights all corresponding circuits, floorplans, and layouts. If the cell is only a component of a visualized floorplan, the corresponding part of the floorplan will be highlighted.

The list-oriented browsers of `PLAYBASE` are integrated into `DESIMA`, too. Until now, they are the most suitable way to navigate through the data model and an excellent tool to show the actual state of the design transactions. `DESIMA` also allows to switch from the browsers to the graphical representation of the entries (design objects) and vice versa at any time.

Interactive design management is a further feature of `DESIMA`. The designer can select any depicted object for a design action (a graphical object as well as an entry in a browser list). This can be a single object or a group of objects. After pressing the “start task” button from a pop-up menu (Figure 32) DESIMA calls PLAYFRAME to create a new design transaction and to retrieve the corresponding toolbox input data for each of the selected design objects. Then the toolbox will be started.

The design flow model is implemented by an extended Petri net [Bre93]. Transitions of the net represent design tools (more precisely: design tasks because some toolboxes are able to perform different tasks or we can combine two or more stand-alone tools to a single task). Design data are described by tokens on the places which means that the places represent data types. A design tool is executable in the case that the corresponding transition of the Petri net is enabled (i.e. it can fire). A planning component based on the Petri net supports the designer in navigating through the net in a suitable manner. While the net itself represents actual design tools and data dependencies, the PLAYOUT planning component is based on estimation tools (e.g. our SHAPE FUNCTION GENERATOR).

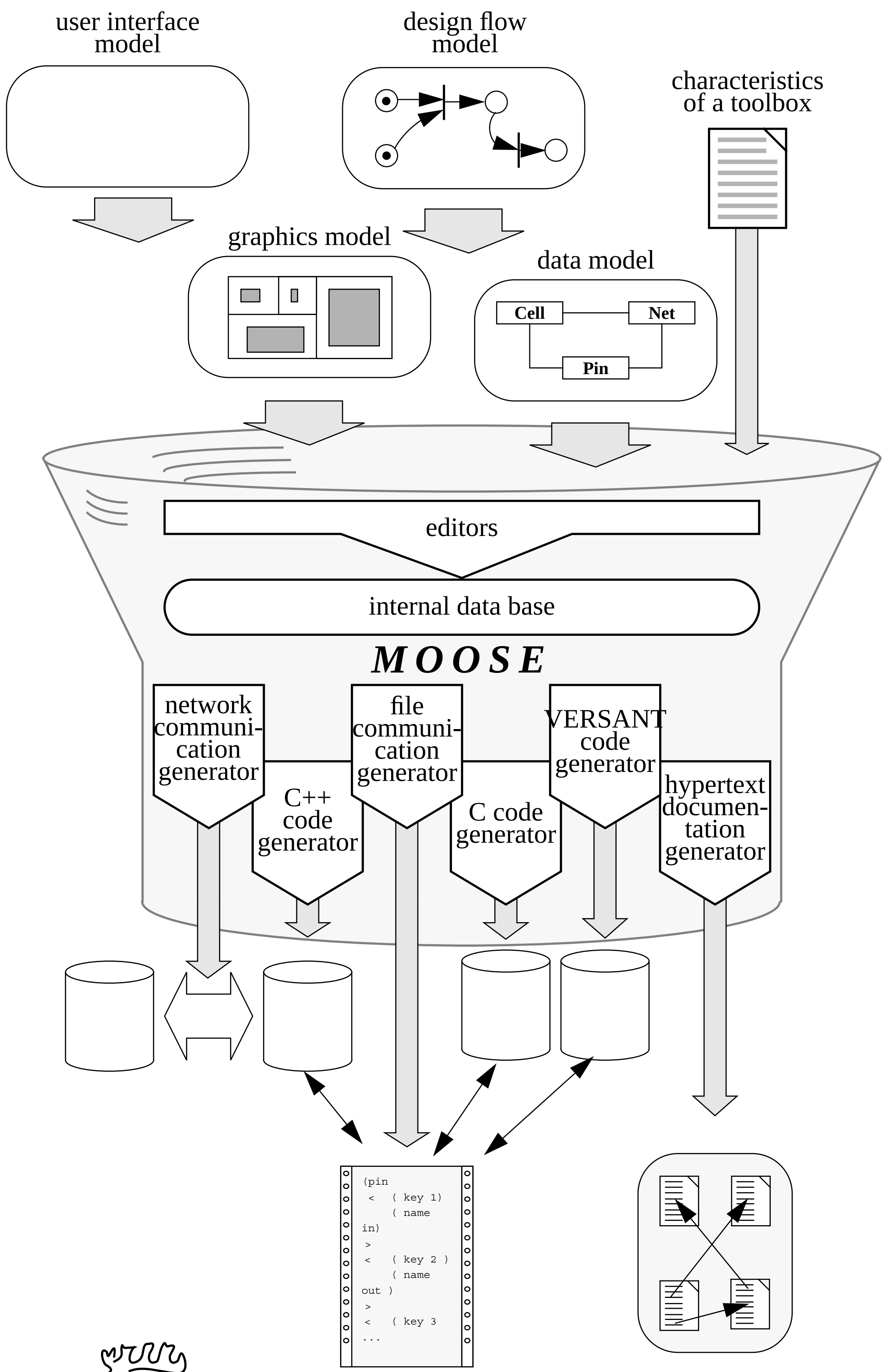
5.3 Generator-Based Toolbox Development

Besides supporting the VLSI design itself by the PLAYBASE data management component and the DESIMA design management component the PLAYOUT CAD framework group also works on the support of the toolbox integration. Our overall approach is the automatic generation of as much of the toolbox' source code as possible. The input of the code generators are abstract models, e.g. extended Entity Relationship diagrams for the data management component [ASS95]. Our ideas resulted in the MOOSE software development environment (MOOSE is an acronym for Model-based Object-Oriented Software generation Environment). Figure 33 depicts an overview of our approach.

The precondition for the success of our generator concept was the fact that the major part of the source code is very similar for all toolboxes of a design system. For instance, most of the toolboxes have a graphical user interface which has to be nearly identical for all toolboxes to guarantee a common look and feel. In addition, the data management component and the control flow component are not very different between the particular toolboxes. It therefore makes sense to implement these components only once and to reuse these implementations of the components for all toolbox. This idea is similar to the library approach.

However, our experiences have shown that a rigid library concept (as it is the case for the X-WINDOWS and OSF-MOTIF libraries) is not suitable for our requirements [Pah94]. Although a component has similar functionality for different toolboxes, there may be different requirements for the runtime behavior, memory consumption, etc. For example, the netlist of a circuit has to be stored in a compact linear list for one toolbox while we prefer a fast hashing structure for another toolbox.

Our idea is not to use a library implementation for reuse but to reuse the functionality of the corresponding component which will be represented by an abstract model. For each particular toolbox implementation this model will be modified or adjusted to meet the individual requirements of the toolbox. Then the modified model together with additional toolbox-specific, non-functional requirements are input to a code generator. MOOSE already contains a large number



```
(pin
 <   ( key 1)
     ( name
in)
 >
 <   ( key 2 )
     ( name
out )
 >
 <   ( key 3
...
```

Figure 33 ***MOOSE***
To reduce the complexity of the figure we only show the code generators of the data management components

of code generators for different framework components and different programming languages (including generators for the database data description language (DDL) and a hypertext documentation generator). By using MOOSE for the toolbox development we saved up to 30% of the programming time. This time can now be better used for finding excellent CAD algorithms.

6. Conclusion

We described a top-down VLSI design method and our design system PLAYOUT. Recently we performed several large test designs to show the quality of this approach. The examples run so far show that such a method is feasible and superior to bottom-up methods. The design of the largest test circuit shall shortly be described on behalf of all other designs.

<u>The XLII test design</u>

Circuits with at least three hierarchy levels are necessary to perform a realistic hierarchical design with two or more planning and one cell synthesis levels. We constructed our own test circuit with approximately 280,000 standard cells because we had no access to a benchmark circuit of this complexity.

We designed four different processing elements (PEs). Each PE was instantiated three times. These twelve PEs were connected to form a single large circuit which we named XLII[5]. Each PE was instantiated three times to conduct several experiments. One of the three instances was configured with many fixed macro cells, resulting in a bottom-up like design strategy. The layouts of the macro cells were synthesized with minimal area and no restrictions to their shapes. The second instance was designed top-down where the sizes and the shapes of the subcells were determined by a top-down chip planning step. The layouts of the subcells must be constructed with respect to these values. The third instance was completed in a mixed manner.

In total, XLII is a chip with twelve PEs which were further partitioned into 40 - 50 standard cell blocks each. After repartitioning XLII was structured into four hierarchy levels:

1. level:	1 chip
2. level:	32 modules (20 flexible cells, 12 memory macro cells)
3. level:	793 standard cell blocks and 60 memory macro cells
4. level:	282,765 standard cells

We used the three configuration types to demonstrate that PLAYOUT can handle different design styles in one design. At all hierarchy levels, the cells consist of fixed and flexible subcells.

After repartitioning we computed the shape functions of all cells bottom-up. We needed two estimation parameter sets. One parameter set was used for the cells for which we generated floorplans assuming that wiring is symmetrical in horizontal and vertical direction. For the standard cell blocks we needed an asymmetrical parameter set that was adapted to the row structure of these cells.

5. The circuit is part of the MCNC benchmark set but is named SALLY there.

After computing the shape function of the top level cell chip planning was started. In the floorplan domain we performed three different experiments:

- pure top-down chip planning
- (iterative) three-phase chip planning
- comparison of the top-down and the bottom-up design styles.

The result of the pure top-down design, i.e. the deviation of the floorplan from the final layout, is shown in the left part of Figure 34. To compensate the variance in the estimated subcell areas we did three-phase chip planning. Since it was not possible to estimate the area of standard cell blocks more precisely than 10 - 15% in all cases, the subcells did not fit accurate into the planned spaces in the floorplan. The empty space increased and so the overall cell area. This could be avoided by the iterative planning method.

cell	**(top-down) floorplan**	**Δ_1**	**layout ***	**layout after iterated planning**	**Δ_2**
PE1.1	280 mm^2	5 %	294 mm^2	290 mm^2	1 %
PE1.2	324 mm^2	4 %	339 mm^2	--- **	--- **
PE1.3	356 mm^2	2 %	349 mm^2	343 mm^2	2 %
PE2.1	285 mm^2	7 %	305 mm^2	--- **	--- **
PE2.2	291 mm^2	17 %	340 mm^2	310 mm^2	10 %
PE2.3	310 mm^2	10 %	341 mm^2	310 mm^2	10 %
PE3.1	347 mm^2	5 %	364 mm^2	359 mm^2	1 %
PE3.2	327 mm^2	10 %	360 mm^2	346 mm^2	4 %
PE3.3	390 mm^2	6 %	413 mm^2	401 mm^2	3 %
PE4.1	233 mm^2	1 %	230 mm^2	--- **	--- **
PE4.2	236 mm^2	2 %	241 mm^2	--- **	--- **
PE4.3	242 mm^2	5 %	254 mm^2	--- **	--- **

*) result of pure top-down design

**) iterative three-phase chip planning has not been performed

Figure 34 Results of the XLII test design
Δ_1: deviations of the floorplans (based on estimated subcells) from layouts
Δ_2: gain of the iterative tree-phase chip planning method.

We tested the iterative planning method in our second experiment. The results are shown in the right part of Figure 34. Figure 35 shows a section of a floorplans at the second hierarchy level as an example. Picture a) shows the floorplan based on the area estimations of the subcells. The second picture shows a revised floorplan that contains the layouts of the subcells.

The layouts were computed with respect to the frames of picture a). While the three cells in the lower-right corner became little smaller than the prediction, the area of the cells on the left-hand side increased. In the iterative planning method, we first inserted the layouts of all critical subcells. After adapting the floorplan of the CUD to the inserted layouts, all other cells were synthesized with a new shape in two further iteration steps. Picture c) shows the result of the iterative planning method. The empty space is much smaller than for the pure top-down method.

a) floorplan with estimated subcells

b) revised floorplan with subcell layouts after pure top-down planning

flexible cell

macro cell

c) revised floorplan with subcell layouts after iterative 3-phase chip planning

Figure 35 Comparison of pure top-down and iterative chip planning

Our third experiment was a comparison between the top-down and the bottom-up approach. As described above, several PEs were configured with many fixed macro cells while other PEs were configured with almost flexible standard cell blocks only. We used the configurations to compare the top-down approach (floorplan with flexible subcells) with the bottom-up approach (floorplan with fixed macro cells). For the experiment we did placement for two cells *PE2* and *PE4* (configured with macros) using the same input frames as for two corresponding cells with flexible subcells (*PE1* and *PE3*, respectively). The result of the comparison is shown in Figure 36. It is conform with our expectation that the placement of macro cells results in a larger empty space (while the top-down chip planning with flexible subcells is less precise because of the inaccuracies of the subcell estimations). We already confirmed this results by further experiments.

cell	configuration	total area	empty space	wiring area
PE1	flexible	310 mm^2	34 mm^2	147 mm^2
PE2	macro	337 mm^2	57 mm^2	157 mm^2
PE3	flexible	346 mm^2	27 mm^2	163 mm^2
PE4	macro	397 mm^2	50 mm^2	205 mm^2

Figure 36 Comparison of top-down and bottom-up design styles

Chip assembly completed the final layout. The layouts of the subcells were composed bottom-up with respect to the top-down floorplan. Using our iterative chip planning method, the rearrangement of the subcells by cell assembly was smaller than the rearrangement of a pure top-down generated floorplan.

Acknowledgment

A system like PLAYOUT is necessary to prove the quality of a VLSI design method. It is also the extent of what a research group in an university can accomplish. Because it is a group effort, the authors names stand in place of many who contributed. Contributions also came from other project groups in the Sonderforschungsbereich (SFB) 124 and from colleagues working in the same field that were guests of the SFB. The SFB is financed by the Deutsche Forschungsgemeinschaft and provided most of the funding. Besides the basic funding by the University of Kaiserslautern, the Zentrum für Rechnergestützte Ingenieurssysteme and the Siemens AG enhanced our funds.

References

[ASS95] Altmeyer, J., Schürmann, B., Schütze, M.. “Generating ECAD Framework Code from Abstract Models”. In “Proc. 32nd Design Automation Conference (DAC)”, San Francisco, June 1995.

[ASZ93] Altmeyer, J., Schürmann, B., Zimmermann, G.. “Three-Phase Chip Planning - Strategy and Flow Control”. SFB 124 report No. 21/93. University of Kaiserslautern, 1993.

[BHN92] Barnes, T.J., Harrison, D., Newton, A.R., Spickelmier, R.L.. “Electronic CAD Frameworks”. Kluwer Academic Publishers, Norwell, MA, 1992.

[Bec93] Becker, T.. “Router-Dependent Estimation of Wiring Area for Chip Assembly”. Master thesis, University of Kaiserslautern, 1993. (in German)

[Bre83] Breuer, M.A.. “A Methodology for Custom VLSI Layout”. Journal IEEE Transactions on Circuits and Systems, Vol. CAS-30, No. 6, 1983.

[Bre93] Bretschneider, F.. “A Process Model for Design Flow Management and Planning”. In “VDI-Fortschrittsberichte, Volume 9, No. 157”. VDI-Verlag, 1993.

[EDI87] “EDIF Electronic Design Interchange Format Version 2 0 0”. Electronics Industries Association, Washington, 1987.

[FiM82] Fiduccia, C.M., Mattheyses, R.M.. “A Linear-Time Heuristic for Improving Network Partitions”. In “Proc. 19th Design Automation Conference (DAC)”, 1982.

[GHZ91] Glasmacher, K., Heß, A., Zimmermann, G.. “A Genetic Algorithm for Global Improvement of Macrocell Layouts”. In “Proc. Int. Conference on Computer Design (ICCD)”, Cambridge, 1991.

[GlZ92] Glasmacher K., Zimmermann G.. “Chip Assembly in the PLAYOUT VLSI Design System”. In “Proc. European Design Automation Conference (EURO-DAC)”, Hamburg, 1992.

[Gro89] Groeneveld, P.. “On Global Wire Ordering for Macro Cell Routing”. In “Proc. 26th Design Automation Conference (DAC)”, 1989.

[HDL88] “IEEE Standard VHDL Language Reference Manual”. The Institute of Electrical and Electronics Engineers, Inc., New York, 1988.

[HNZ93] Hebgen, W., Nuhn, P., Zimmermann, G.. “RETO, an Optimal Clocking Algorithm for Digital Synchronous Circuits”. SFB 124 report No. 42/93. University of Kaiserslautern, 1993.

[HaR85] Hayes-Roth. “A Blackboard Architecture for Control”. Journal Artificial Intelligence, No. 26, 1985.

[KBK90] Kedem, G., Brglez, F., Kozminski K.. “ASIC Design with OASIS”. In “Proc. Int. Symposium on Circuits and Systems”, New Orleans, 1990.

[Len90] Lengauer, T.. “Combinatorial Algorithms for Integrated Circuit Layout”. John-Wiley & Sons, Chichester, 1990.

[MBE90] Masson, C., Barbier, D., Escassut, R., Winer, D., Chevallier, G. F., Zeegers, P. A., Bull, S.. “CHEOPS: An Integrated VLSI Floor Planning and Chip Assembly System Implemented in Object Oriented LISP”. In “Proc. European Design Automation Conference (EDAC)”, Glasgow, Scotland, 1990.

[Mar90] Marwedel, P.. “A Software-System for the Synthesis of Computer Architectures and for the Generation of Microcode”. Habilitation thesis, reprinted as: Report No. 356, Computer Science Institute, Dortmund. University of Kiel, 1990. (in German)

[Mei91] Meixner, G.. “Global Routing of Very Large Scale Integrated Circuits Based on Flow Methods and Timing Considerations”. PhD thesis, University of Kaiserslautern, 1991.

[Ott83] Otten, R.H.J.M.. “Efficient Floorplan Optimization”. In “Proc. Int. Conference on Computer Design (ICCD)”, 1983.

[PDM90] Pedram, M., Dai, W. W.-M., Marek-Sadowska, M., Ogawa, Y.. “Ongoing Research and Development of BEAR Layout System”. In “Proc. Int. Workshop on Layout Synthesis”, Research Triangle Park, NC, 1990.

[Pah94] Pahle, H.. “Model-Based Generation of CAD Framework Components”. PhD thesis, University of Kaiserslautern, 1994. (in German)

[Que94] Queins, S.. “DESIMA, an Interactive Design Management System in PLAYOUT”. Master thesis, University of Kaiserslautern, 1994. (in German)

[RRZ84] Rao, P., Ramnarayan, R., Zimmermann, G.. “SPIDER, A Chip Planner for ISL Technology”. In “Proc. 21st Design Automation Conference (DAC)”, 1984.

[SAS94] Schürmann, B., Altmeyer, J. Schütze, M.. “On Modeling Top-Down VLSI Design”. In “Proc. Int. Conference on Computer-Aided Design (ICCAD)”, San Jose, CA, 1994.

[SAZ92] Schürmann, B., Altmeyer, J., Zimmermann, G.. “Three-Phase Chip Planning - An Improved Top-Down Chip Planning Strategy”. In “Proc. Int. Conference of Computer Aided Design (ICCAD)”, Santa Clara, CA, 1992.

[SeL87] Sechen, C., Lee, K.. “An Improved Simulated Annealing Algorithm for Row-Based Placement”. In “Proc. Int. Conference on Computer-Aided Design”, Santa Clara, CA, 1987.

[SiZ89] Siepmann, E., Zimmermann, G.. “An Object-Oriented Datamodel for the VLSI Design System PLAYOUT”. In “Proc. 26th Design Automation Conference (DAC)”, Las Vegas, 1989.

[Sie88] Siepmann, E.. “PLIF - an Object-Oriented Data Exchange Format for the Communication in PLAYOUT”. SFB 124 report No. 32/88. University of Kaiserslautern, 1988. (in German)

[Sie89] Siepmann, E.. “A Data Management Interface as Part of the Framework of an Integrated VLSI-Design System”. In “Proc. Int. Conference on Computer-Aided Design (ICCAD)”, Santa Clara, CA, 1989.

[SpS88] Spang, H., Siepmann, E. . “An Object-Oriented Toolbox Manager for the VLSI Design System PLAYOUT”. ZRI report No. 5/88, University of Kaiserslautern, 1988.

[SuA95] Schürmann, B., Altmeyer, J.. “The Effect of Pin Constraints on Layout Area”. In “Proc. IEEE European Design and Test Conference (ED&TC)”, Paris, 1995.

[SzO80] Szepieniec, A.A., Otten, R.H.J.M.. “The Genealogical Approach to the Layout Problem”. In “Proc. 17th Design Automation Conference (DAC)”, 1980.

[YiW89] Ying, C.-S., Wong, J.S.-L.. “An Analytical Approach to Floorplanning for Hierarchical Building Blocks”. Journal Transactions on Computer-Aided Design, 1989.

[Zim81] Zimmermann, G.. “CAD Tools for the Synthesis of Hardware and Software”. In “Proc. 5th Intern. Conf. on Computer Hardware Description Languages ”, Kaiserslautern, 1981.

[Zim88] Zimmermann, G.. “A New Area Shape Function Estimation Technique for VLSI Layouts”. In “Proc. 25th Design Automation Conference (DAC)”, Anaheim, 1988.

[Zim88.1] Zimmermann, G.. “The MIMOLA Design System - A Computer Aided Digital Processor Design Method”. In “25 Years of Electronic Design Automation, A compendium of papers from the Design Automation Conference”, 1988.