

# Java-Based Learning of Algorithms for VLSI Physical Design Automation\*

SERAFÍN A. PÉREZ-LÓPEZ, MARÍA JOSÉ GONZÁLEZ-BRAÑA, JORGE MARCOS-ACEVEDO, MARÍA DOLORES VALDÉS and ENRIQUE MANDADO

*Departamento de Tecnología Electrónica e Instituto de Electrónica Aplicada 'Pedro Barrié de la Maza', University of Vigo, Spain. E-mail: sperez@uvigo.es*

*Courseware is presented here, based on Java applets, that is designed to facilitate better understanding of some of the most popular algorithms in semi-custom IC physical design automation. The applets have been applied successfully in the course Microelectronic Design II, a telecommunication engineering fifth-year subject at the University of Vigo (Spain). This method does not wholly replace traditional lecture classes, but it enhances them by displaying graphically the intermediate solutions of an algorithm's execution. Thus, students can easily and quickly assimilate what implies a change of value in a certain design parameter, such as the annealing temperature, the crossover or mutation operators, the cost function of a placement, a routing, etc. In fact, they not only understood much more rapidly all those concepts inherited from nature, but they repeatedly enjoyed executing the algorithms by changing their parameters, with one criterion in mind (not randomly), in order to optimize a given layout.*

## INTRODUCTION

EVER SINCE VLSI started in the 1970s, micro-electronic physical design has developed extensively, in such a way that today it is possible to fabricate integrated circuits containing millions of transistors. This progress would not have been feasible without sophisticated software design tools, which must not only be computationally fast but also able to obtain results that are close to optimal because all phases of the physical design constitute NP-complete problems; that is, it is not possible to get an optimal solution in a reasonable amount of computing time. The most powerful computer or computer network may need many years to explore exhaustively all possible cases.

During the last three decades, investment in Physical Design Automation has been intensive, as evidenced by the large number of research articles published and CAD tools developed for this final stage of circuit design [1–3]. Therefore, it is difficult to assess, classify and eventually teach the main algorithms, concepts and solutions proposed since then.

This work is intended to help students learn the main physical design automation algorithms, not simply by exposing the operation sequence and pseudo-code (classical method), but mainly by graphically seeing its evolution in order to achieve an optimal layout partitioning, placement and routing in a Java-based environment (proposed method). Java applets have been used in order to obtain platform independence and user interaction.

Commercial tools compute solutions by showing

a small amount of the information about the algorithm's evolution, but they do not usually display the intermediate solutions, which may be interesting from a didactic perspective for a better understanding of the algorithm. Filling this gap is the main goal of the applets developed for this course.

Thus, the courseware provides a learning environment that augments the traditional one and helps motivate students, avoiding them just passively viewing a pseudo-code. It also allows learning by example, which is most students' preferred method of learning. Java allows for the development of interactive examples, so they can be asked to predict what would happen when a parameter or certain conditions are changed. Our experience has shown that this courseware yields both high student motivation and good learning results.

We divided the process into several parts, each one related to a specific phase in the physical design. They are, however, linked, so that the results from one act as input to the next. For instance, the placement obtained by applying a simulated annealing algorithm is the initial information for the global routing algorithms. In this way, the intention is to approach the real design process, whereby a problem passes through all phases until it reaches a satisfactory solution.

In the next section of this paper, the physical design phase for semi-custom IC is briefly introduced. Furthermore, an analysis is made on how new trends affect the design cycle. The following section outlines some of the applets developed and shows their capabilities. Following this, the simulated annealing (SA) and simulated evolution (SE) algorithms applied to the placement and routing

\* Accepted 23 September 2004.

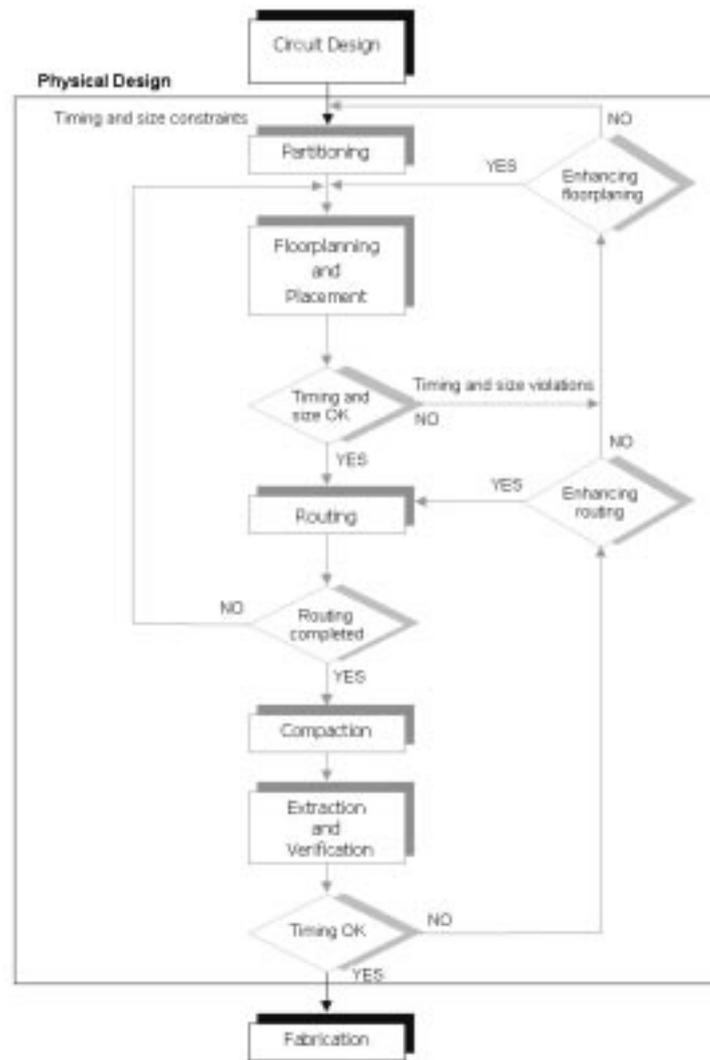


Fig. 1. Physical design cycle.

phases are explored. We selected both methods because they are so well accepted, especially SA. SE algorithms have been implemented less in commercial EDA tools due to their inefficiency in terms of runtime. However, this may change in the future as new hardware and software platforms and tools become increasingly more powerful. In that case SE could be used more due to its more exhaustive searching capability.

This paper ends with a few words on the conclusions of our research and our plans to extend and enhance this tool in the near future.

#### AN OVERVIEW OF SEMI-CUSTOM IC PHYSICAL DESIGN

As can be seen in Fig. 1, VLSI physical design basically comprises six stages: partitioning, floorplanning, placement, routing, compaction, and extraction and verification.

Because of memory and processing limitations, it is not possible to directly obtain a good layout of

a large circuit, so the solution usually adopted is to divide the circuit into blocks. This process is called *partitioning*. In the partitioning process, factors such as block count, number of interconnections between them or their sizes have to be considered.

The *floorplanning* phase comprises the selection of a suitable layout for each block. The block area must be estimated and the shape selected. The area estimation depends on the block count, the block types and the interconnection area. Blocks are often rectangular but in some cases can have other shapes, except curved sides.

In the *placement* phase, each block is assigned to an exact position in the available area. The objective is to obtain a block arrangement of a minimum area that allows all connections to be made, while meeting some given constraints. Usually, an initial placement is obtained and iterative improvements are made until the layout shrinks or has a better ability to meet the design specifications. In addition, some space has to be left empty to allow interconnections between blocks. The quality of the placement will not be evident until the routing

phase has been completed. What is more, the routing may not be feasible in the available space, in which case another placement iteration is required.

The goal of the *routing* phase is to carry out the interconnections between blocks according to the given netlist. Firstly, the space not occupied by the blocks is partitioned into rectangular areas called channels and switchboxes. The router tries to complete all connections, minimizing wire length. Generally this is done in two steps: global routing and detailed routing. The former finds a list of channels and switchboxes for each wire. Detailed routing is done for every channel or switchbox separately, calculating the coordinates in order to place each connection segment.

*Compaction* consists of compressing the layout in all directions so that the whole area shrinks. This also implies a reduction in wire length and signal delay. At the same time, more chips can be produced on a wafer, which in turn reduces the cost of manufacturing.

Design Rule Checker (DRC) verifies that all geometric patterns meet the design rules imposed by the fabrication process. After removing the design rule violations, the layout must be verified through circuit *extraction*. This is a reverse engineering process that allows the schematic to be generated from the layout. Then the extracted description is compared with the original circuit for *verification* of its correctness. This process is

called ‘layout versus schematics’ (LVD) verification.

As fabrication technology improves and processes enter the deep sub-micron range, interconnect delay is not scaling at the same rate as gate delay. As it becomes a more significant part of the overall delay, in high performance chips, it must be considered from the very early design stages. Several methods can be employed to reduce it, such as:

*Chip level signal planning*: At the chip level, the routing of the main signal and buses must be planned from the early design stages, so that interconnection distances can be minimized. In addition, these global signals must be routed in the top-level metal (less parasitic capacitance) to minimize delay per unit length.

*OTC routing*: Over-the-cell (OTC) routing refers to the routing made over blocks and active areas. This is a departure from the conventional channel and switchbox routing approach. Chip level signal planning is in fact OTC routing on the entire chip. This approach essentially makes routing a three-dimensional problem. Another effect of it is that pins are not brought to the block boundaries for connection to other blocks. Instead, pins are brought to the top of the block as a sea-of-pins. This concept is called the ‘arbitrary terminal model’ (ATM).



Fig. 2. Menu for the placement phase.

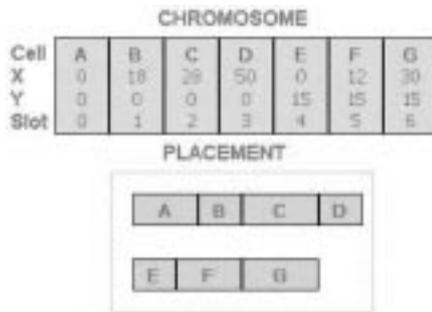


Fig. 3. A cell placement and its genetic representation.

This research has not considered the new trends so far, but we intend to do so in the near future.

### DEVELOPED APPLETS

So far our applets comprise three main phases in the VLSI physical design: partitioning, placement and routing (both global and detailed). For this paper, we only describe some of the applets developed for placement and routing. All of them can be viewed at: <http://www.dte.uvigo.es/articulos/al4pda>.

Three applets were developed for the placement phase:

Genetic algorithm for standard-cell placement (GASP)

Simulated annealing (SA)

Simulated annealing and genetic algorithm for standard-cell placement (SAGA)

One was developed for global routing:

Maze routing

And one more was developed for detailed routing:

Greedy detailed routing algorithm

As can be seen in Fig. 2, each window, in the rightmost frame (and scrolling down the page), presents a general description of the selected algorithm, focusing on:

Main concepts and features

Working parameters

Target function to minimize

Specifying the input data

Ancillary windows for easily following the execution

How the results are shown and should be interpreted

### PLACEMENT

Clicking on the *Placement* button, the window in Fig. 2 will appear with the options already mentioned: GASP, SAGA and SA.

GASP [3] is a placement algorithm designed according to the simulated evolution concepts introduced by Holland [3] in 1975. Each placement



Fig. 4. Library cell used in the GASP placement example.

solution (chromosome or individual) is represented, as shown in Fig. 3. That is, each gene (corresponding to a cell in the placement) consists of four fields:

Cell identifier

X-coordinate of the cell upper left corner

Y-coordinate of the same point

Identifier of the slot associated with the cell

Each chromosome's fitness is given by the equation (1):

$$f = \frac{1}{\sum_{nets} [x(i)W_H(i) + y(i)W_V(i)]}$$

where  $x(i)$  and  $y(i)$  are, respectively, the width and height of the lowest rectangle including all pins of a given net  $i$ .  $W_H(i)$  and  $W_V(i)$  represent the width and height of assigned weights. This expression does not include overlapping penalization, as most algorithms do, because overlaps are removed before fitness evaluation.

Four basic genetic operators are used: selection, crossover, mutation and inversion.

A random initial population is generated, in which every individual has an associated cost according to (1), and then an iteration takes place (applying the mentioned operators) to generate some offspring. The new population must be reduced in order to hold constant the individual count. The probability of each individual surviving is a function of its fitness, so the population improves at each iteration or age. When a given age count is reached, the best individual becomes the solution.

The implementation done finds a solution to a placement of 75 cells belonging to the library shown in Fig. 4. The upper pins are inputs and the lower are the cell outputs. The circuit netlist is generated randomly. An available area of  $400 \times 440$  units has been considered.

Four windows are used to enter the data and easily follow the placement evolution:

*Control.* This window allows the user to select the working parameters, as can be easily seen in Fig. 5a.

*Algorithm evolution,* which appears only during algorithm execution to show the actual execution percentage and generation (see Fig. 5b).

*Placement display window.* This shows the placement of the best individual in the population (Fig. 6 shows an example). The display-refreshing period can be selected. A color code (which obviously cannot be distinguished in this black-and-white figure) indicates the longest nets



Fig. 5. GASP algorithm control and evolution windows.

(more than 700 units, in red), the intermediate (less than 700 and more than 400, in blue) and the shortest ones (less than 400 units, in gray).

*Cost evolution*, which shows the population and the best individual average cost evolution as the algorithm is executed between generations. Figure 7 displays two graphics of the same GASP example with different values of population size and generation count. As generations evolve, both costs shrink.

The SAGA algorithm is a combination of genetic and simulated annealing placement, which improves the convergence process due to the initial fast convergence of the genetic placement and the final one in the case of the annealing. For further details about the algorithm, references [3] and [4] can be consulted.

The implementation done finds a solution to the same problem shown above for GASP. When the algorithm finishes, some feedthrough cells are inserted if indicated by the user in the control panel, with a capacity of 16 nets passing through. The number of feedthroughs inserted into each row depends on its width.

Again, four windows are used to enter the data and easily follow the placement evolution: control (see Fig. 8), cost evolution (Fig. 9), placement display (Fig. 10) and algorithm evolution (similar to Fig. 7).

The cost evolution window, which only appears during algorithm execution, indicates the current phase:

*Genetic evolution*: The execution percentage is displayed as well as the actual generation count.

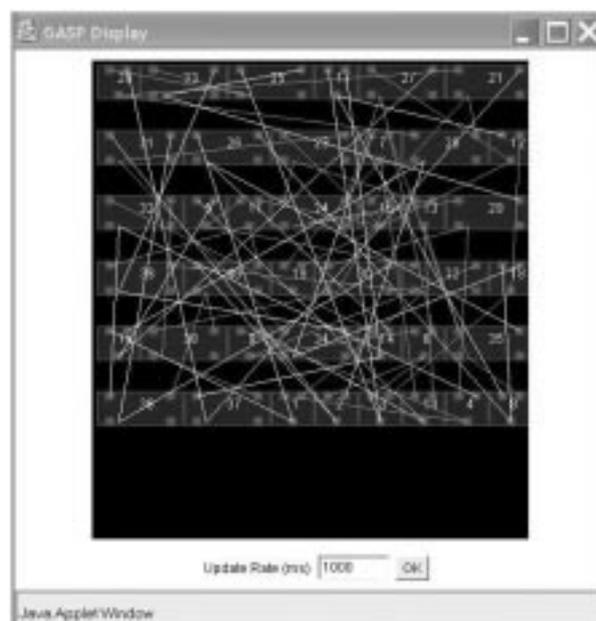


Fig. 6. GASP placement display window.

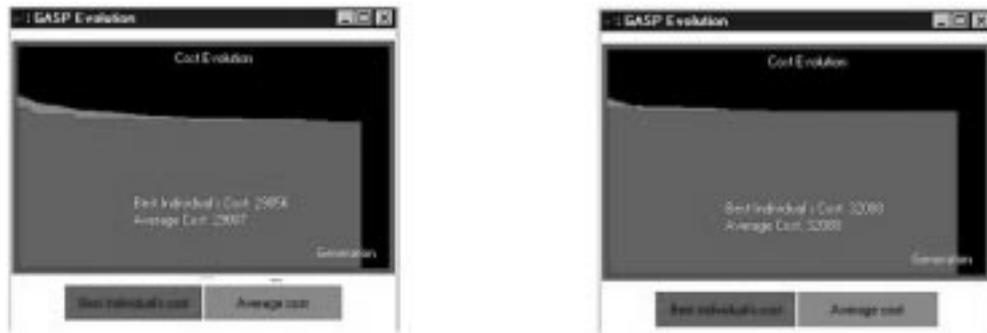


Fig. 7. GASP cost evolution window.



Fig. 8. SAGA algorithm control window.

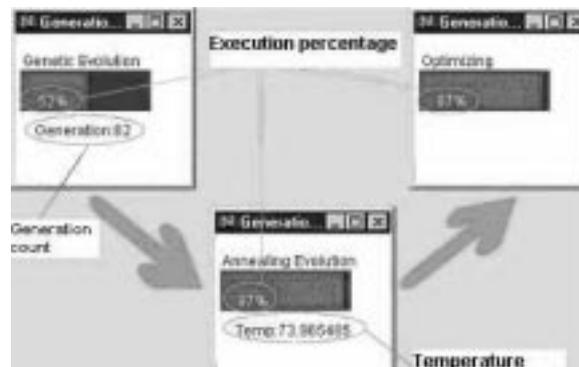


Fig. 9. SAGA cost evolution window.

*Simulated annealing:* This is executed when the population is just one individual. The execution percentage and the actual individual temperature are displayed.

*Optimization:* This is the final step. The window shows the algorithm execution percentage.

The placement display window shows the placement associated with the current best individual. Again, the user can select the display-updating interval. Furthermore, by clicking the right mouse button on the desired cell, this and the nets connected to its output terminals change to orange. In the same way as GASP, each net is colored according to its length. Feedthroughs are represented by yellow rectangles in each row.

The algorithm evolution window allows the current population average cost evolution to be

followed, as well as the best individual cost recorded.

Regarding the SA algorithm, two versions have been implemented: Jespen-Gelatt [5] and Timber Wolf [6].

#### JESPEN-GELATT

A placement solution is given by the whole module locations in the chip as well as the rotations and reflections, with respect to a standard disposition that they have experimented. It has been assumed that all modules are rectangular with different proportions and sizes. The algorithm allows overlapping in the intermediate configurations, which simplifies the definition of the three possible movements: M1 (module shifting), M2 (90°

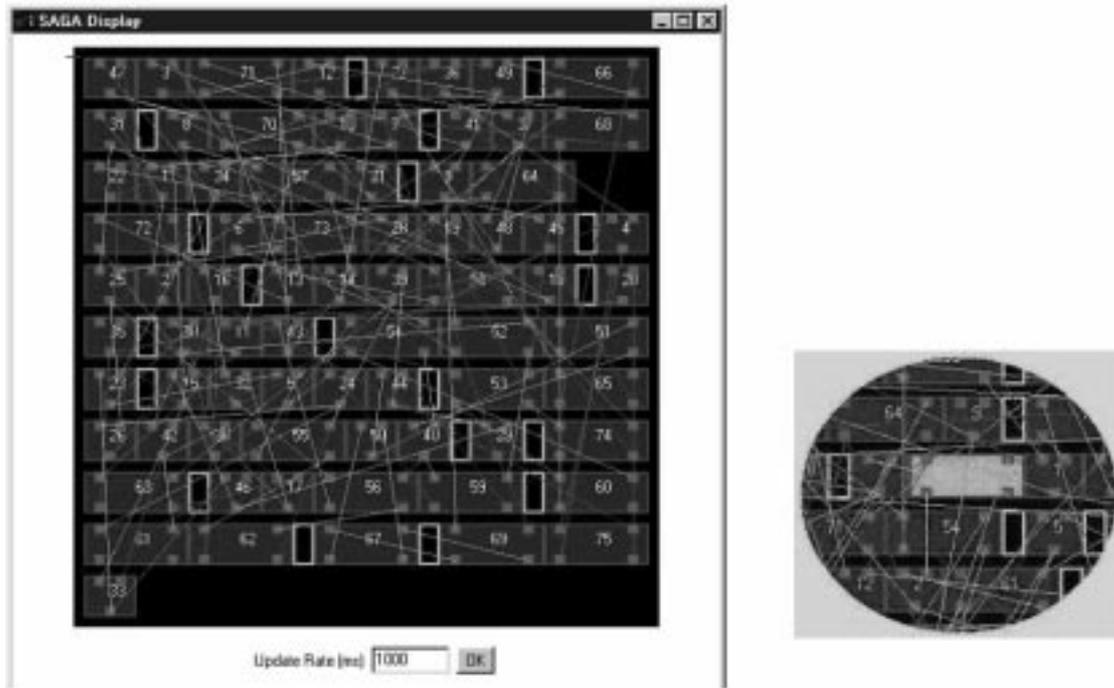


Fig. 10. SAGA placement display window and a local zoom in.

rotation) and M3 (module reflection). The cost function is given by the sum of two terms:

- **W**: interconnection cost estimation. With a superimposed grid, the cost is estimated by the number of nets passing through each grid line and the sum of the cell overlapping lengths with the grid lines.
- **O**: penalization function for module overlapping, given by the sum of the squares of the cell overlapping areas, weighted by a factor  $r$  ranging from 0 to 1.

#### TIMBER WOLF

This algorithm is similar to the last one. The differences are that Timber Wolf allows some flexibility in module shaping, which may be useful for minimizing the total chip area, and it also allows the modules to have mobile terminals. The possible movements are: M1, M2 and M3 (idem to Jespen-Gelatt), M4 (change in the aspect ratio of a module) and M5 (change pin site assignment of a pin or a group of them). The cost function is the sum of three terms:

An estimation of the interconnect length, given by the sum of the rectangular half-perimeters associated with the nets.

An overlapping penalization given by the sum of the squares of the cell overlapping areas.

A penalization due to the pin site capacity overflowing.

The implementation done allows the user to generate a random proof case with a given area ratio, which can be selected in the control panel. This parameter is the ratio between the area occupied

by the cells and the available placement area. If it takes small values, the generated input case will be simple: there will only be a few cells and overlapping will be eliminated quickly. On the other hand, if its value is high, the input case will comprise many cells and, thus, the convergence process and elimination of overlapping will be slower. There are two steps in the generation process of an input case:

**Cell generation.** Cells (belonging to any of the nine basic types, Fig. 11) are randomly generated until the area is full. They are all assumed to have their input pins in the upper and left sides, and the outputs in the lower and right sides. Figure 12 shows the available cell library, whose numbers refer to Fig. 11.

The cells are interconnected randomly according to the next process: an output pin is connected (with a probability of 0.9) to another randomly generated input. With a probability of 0.1, the output is connected to two randomly generated inputs. This means that there will be both two-terminal and three-terminal nets. This process goes on until all outputs have been connected or all inputs have already been assigned. It may happen that some outputs or inputs remain unconnected. In this case, each free input terminal will be connected to a randomly generated output, thus obtaining nets with more than three terminals.

Once the netlist is generated, locations are assigned, also randomly, to the cells (overlapping is allowed by now) and then the placement algorithm (Jespen-Gelatt or a simplified version of Timber Wolf) is executed.

Type	Width	Height	Inputs	Outputs
1	24	24	2	2
2	24	40	3	3
3	24	72	5	5
4	40	24	3	3
5	40	40	4	4
6	40	72	6	6
7	72	24	5	5
8	72	40	6	6
9	72	72	8	8

Fig. 11. Cell types for the SA macrocell placement.

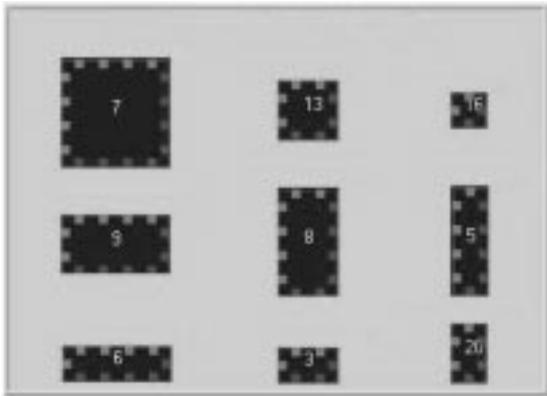


Fig. 12. Library cell for the SA macrocell placement.

Four panels are used for the program interface:

1. *The control panel* (see Fig. 13a). Either of the two algorithms can be selected. The other options are:  
 Randomize. A new random configuration of the current cell list is generated and displayed.  
 Run SA. The algorithm starts execution.  
 To. Initial temperature selected.  
 Tfinal. T at which the algorithm finishes execution.

New case. A new cell list is generated such that the sum of their areas is a percentage (area ratio) of the total available area for the layout ( $400 \times 400$ ).

Iterations. Iteration count at each temperature.  
 Alpha. Weight of penalization overlapping.

Beta. Associated weight to the sum of the cell lengths overlapping to line grid (only for Jespen-Gelatt).

Area ratio. Percentage of the total area occupied by the cells.

Store/Load IN. Storing/retrieving an input case (cell list, netlist and cell configuration).

Store/Load OUT. Storing/retrieving an algorithm result for an input case.

2. *Algorithm evolution* (Fig. 13b, active only during execution). This displays the percentage of execution and current temperature.
3. *Placement display* (Fig. 14). This panel shows the configuration currently being processed. Once again, a color code is employed to distinguish the different net lengths as well as the inputs from the outputs. Furthermore, there are two buttons: OK.
  - The value introduced into the text field becomes the current screen refreshing time for the layout.
  - GRID. It can be seen that the grid employed by the Jesper-Gelatt algorithm. To eliminate it, simply click on the button again.
4. *Cost function evolution* (Fig. 15). This shows the values of both terms in the cost function during execution. For each one, the highest value taken is also displayed.

### GLOBAL ROUTING PHASE

As stated above, the goal of this phase is to obtain the channel list to form the path between each source and target. To access the global routing, one must return to the main menu and

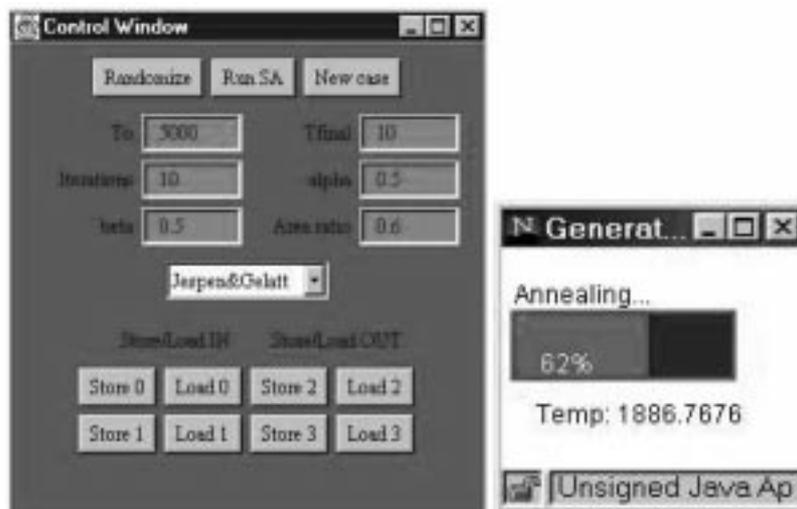


Fig. 13. Control and algorithm evolution panels for SA macrocell placement.

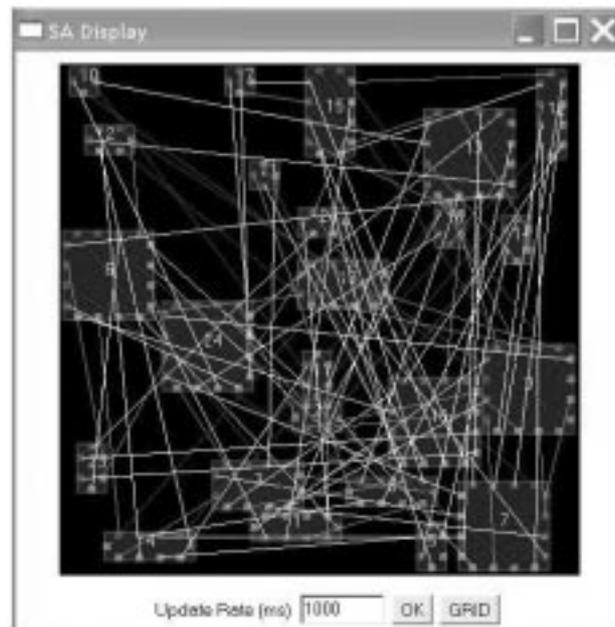


Fig. 14. Placement display panel.

select the *Routing* option to bring up the window in Fig. 16. If the user clicks on the OPEN button (just below the text '(2) Routing'), global routing starts. If the placement is not yet completed, the option '(1) Placement' allows the user to return to this phase.

In 1961, Lee presented an algorithm for routing two terminal nets on a grid. There have been many improvements since then, mainly in processing speed and memory requirements, which has led to the maze routing algorithms. These represent the available routing areas as unblocked vertices and the obstacles as blocked. A path between the source and target vertex must be found without using any blocked ones. The process consists of two phases:

*Exploration:* Several paths start at the source and expand until one reaches the target.

*Retrace:* When the target is reached, the vertices must backtrack to the source to identify the

path. This can be easily implemented as long as the information on the parentage of each vertex is kept during the exploration phase.

The implementation again uses four panels:

*Control.* The user can move between the placement and routing applets, as well as select the maze routing he wishes to execute.

*Get placement.* The solution generated by the placement phase is the input to the routing algorithm.

*Run global routing.* The maze algorithm is applied to that placement.

*Cost evolution panel.* This window (similar to Fig. 5b) shows a progression bar with the percentage of execution, the number of routed nets and the time employed in routing the last net.

*Routing display.* The routing evolution is displayed while the algorithm is executed, representing the last routed nets in magenta and the ones already routed in yellow. Figure 17 shows a detail of a placement obtained by a SAGA algorithm. The selected cell and its output nets are highlighted in orange to better see their routing.

*Result display panel.* When execution ends, a new window displays the routing cost obtained, given by the sum of all net lengths (given in turn by the slot count) and routing runtime.

The 'Greedy' option (see Fig. 16) must be selected to enter this phase, thus bringing up the window shown in Fig. 18. An optimal routing can be obtained if, for every column, there is just one track per net. Greedy's algorithm [7] is based on that observation. Basically, it tries to reduce the channel height by routing the nets column by column and placing the horizontal trunks, belonging to the same net, as near as possible. Dogleg insertion is allowed in any column.



Fig. 15. Cost function evolution panel.

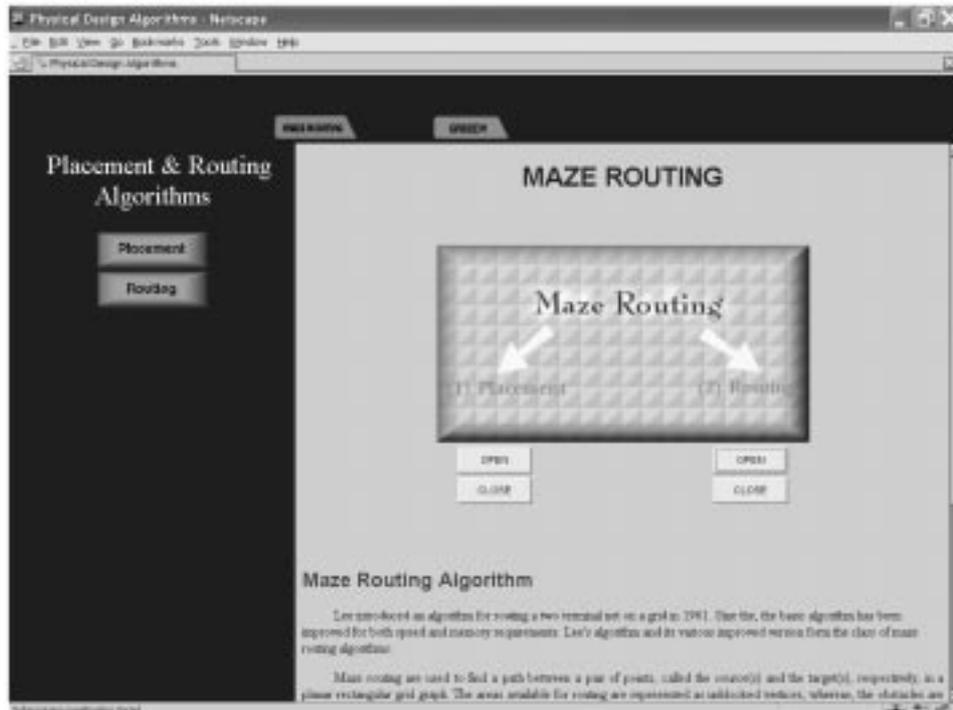


Fig. 16. Routing main menu.

### DETAILED ROUTING PHASE

To execute the algorithm, the placement and global routing algorithms must be executed first. Clicking on the OPEN button, two panels will appear on the screen. During execution, an addi-

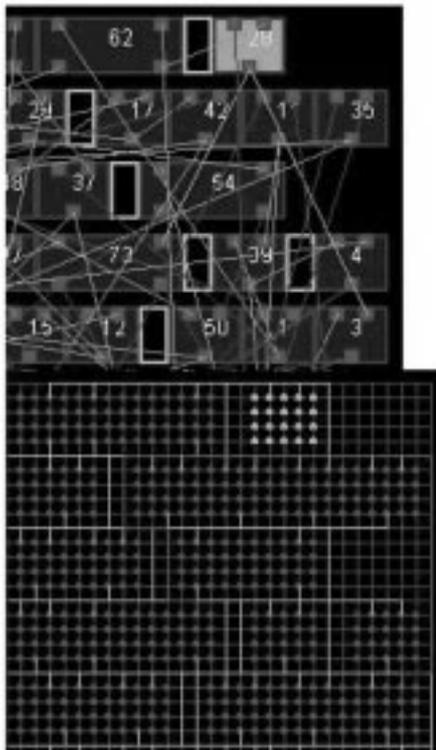


Fig. 17. Global routing display panel.

tional one shows the execution state. These panels are:

- *Control*. One option captures the global routing result and another one starts the execution.
- *Algorithm evolution*. Similar to Fig. 5b.
- *Routed channel and completed channel display*. The purpose of this is twofold:
  1. To display every channel routing. According to the implementation done, there are always 13 or 14 channels in the layout (11 or 12 horizontal and 2 vertical). Numeration starts from the upper horizontal channel (0) to the lower one (10 or 11). The left vertical channel is the 12th or 13th and the right one the 13th or 14th. Thus, the channels are displayed sequentially in the order described, showing number and density at the top of the window.
  2. To display a local zoom-in of the routed circuit (Fig. 19). Any area in the layout can be selected (by using the scrolling and shifting bars) to zoom in.

Finally, Fig. 20 shows a completely routed channel in more detail.

### CONCLUSIONS

We have developed a tool to help teach some of the major physical design automation algorithms, mainly based on graphically displaying their evolution to achieve an optimal layout partitioning, placement and routing. Java applets have been used in order to get platform independence and, above all, user interaction.

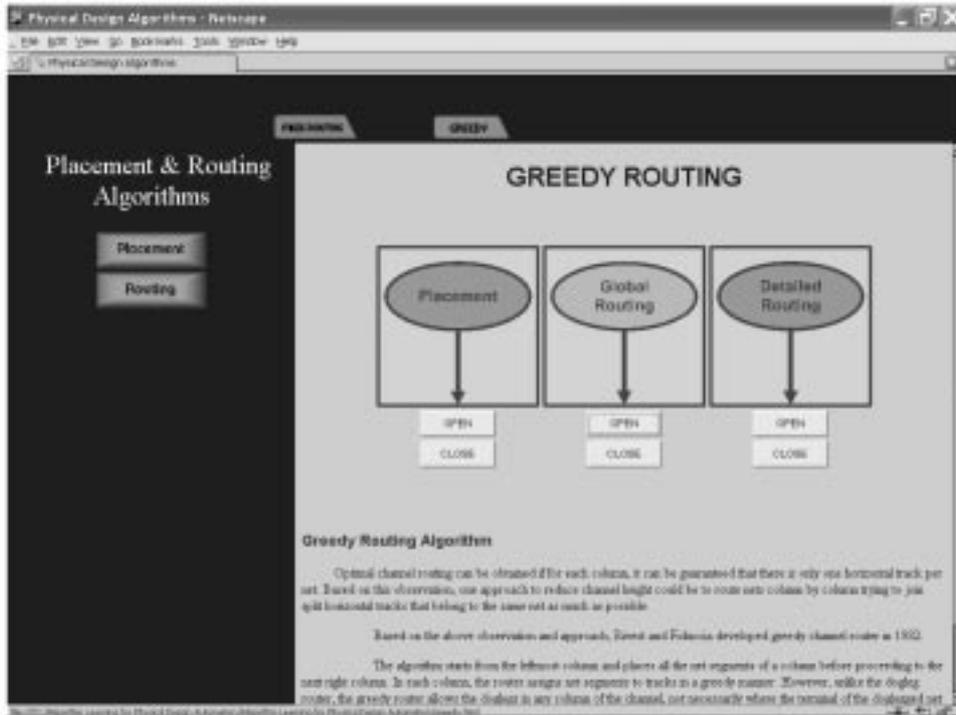


Fig. 18. Detailed routing menu.

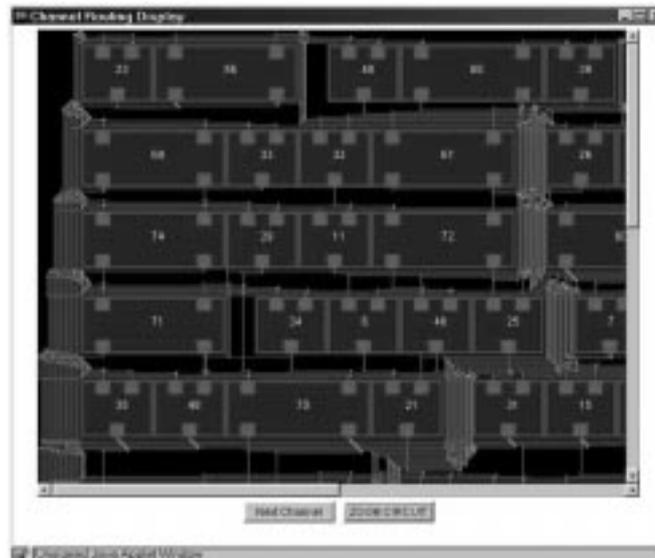


Fig. 19. A zoom-out view of several routed channels.

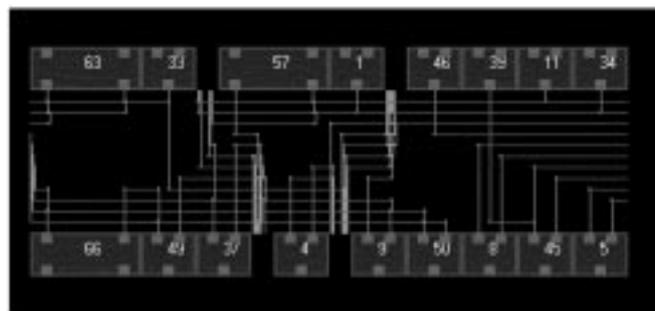


Fig. 20. A more detailed view of a completely routed channel.

The different sections are accessed through a web page. Each one corresponds to a phase in the semi-custom physical design and implements some selected algorithms as well as an introduction to them.

The didactic approach prevails through all this courseware, not just by showing in detail the graphic evolution, but also by the possibility of interactively changing any working parameter to

better understand its effect in the algorithm searching and the layout quality.

Student feedback has shown a high level of acceptance and they were clearly much more interested in physical design than their fellows in previous years, so we are currently trying to improve the courseware. We will be adding some more items, such as floorplanning and compaction, as well as algorithms for high performance driven partitioning and OTC routing.

## REFERENCES

1. N. Sherwani, *Algorithms for VLSI Physical Design Automation* (3th edition), Kluwer Academic Publishers (1999).
2. D. F. Wong, H. W. Leong and C. L. Liu, *Simulated Annealing for VLSI Design*, Kluwer Academic Publishers (1988).
3. P. Mazumder and E. M. Rudnick, *Genetic Algorithms for VLSI Design, Layout and Test Automation*, Prentice-Hall International (1999).
4. H. Esbensen and P. Mazumder, SAGA: Unification of genetic algorithm with simulated annealing and its application to macro-cell placement, Proceedings of IEEE International Conference on VLSI Design (January 1994).
5. D. W. Jespen and C. D. Gelatt, Macro placement by Monte Carlo annealing, Proceedings of IEEE International Conference on Computer-Aided Design (1983), pp. 495-498.
6. C. Sechen, *VLSI Placement and Global Routing using Simulated Annealing*, Kluwer Academic Publishers (1997).
7. R. Rivest and C. Fiduccia, A greedy channel router, Proceedings of 19th ACM/IEEE Design Automation Conference (1982), pp. 418-424.

**Serafin A. Pérez-López** received his degree in Telecommunications Engineering from the Polytechnic University of Catalonia (1984) and his Ph.D. from the University of Vigo (1991). He has been a Titular Professor in Electronic Engineering at this university since 1992. His research and development works comprise ASIC design and microcontrollers, DSP and FPGA application design based on the standardized hardware description languages (VHDL and Verilog).

**María José González-Braña** received her degree in Telecommunications Engineering in the specialized field of Electronics from the University of Vigo in October 2002. She was Student-Researcher Period at IMEC (Interuniversity Microelectronics Center) from January 2002 to July 2002, working in MICS (Multimedia Image Compression Systems). She is currently a researcher and Associated Professor in the Department of Electronic Technology in the University of Vigo.

**Jorge Marcos-Acevedo** received a Ph.D. degree in Industrial Engineering in 1994 from the University of Vigo (Spain). From 1983 to 1988, he worked as an engineer of instrumentation in the maintenance-process control department in Alúmina-Aluminio. Since 1998, he has been a professor at the University of Vigo. His current research is on power electronics (fast battery chargers), electrolyte sensors in lead-acid batteries for electric vehicles and reliability and safety in electronics systems.

**María Dolores Valdés** received a degree in Electrical Engineering (1990) from the Central University of Las Villas (Cuba) and a Ph.D. in Telecommunication Engineering (1997) from the University of Vigo (Spain). Nowadays she is a visiting professor in the Department of Electronics Technology of the University of Vigo. Her current research interest is in the area of reconfigurable logic for virtual instrumentation and DSP systems.

**Enrique Mandado** received a B.Sc. degree in Electronic Engineering (1969) from the Polytechnic University of Madrid, and a Ph.D. from the Polytechnic University of Barcelona (1976). From 1969 to 1978 he was an electronic application engineer working for Philips. In 1979 he moved to the University of Vigo where he is now a Professor of Electronics and Director of the Institute for Applied Electronics, developing electronic systems for control and measurement. He has published ten books about electronics including *Sistemas Electrónicos Digitales* (now in its eighth edition) (Marcombo) and *Programmable Logic Controllers and Logic Devices* (Prentice-Hall).