Multithreaded Parallel VLSI Leaf Cell Generator Using Agents 2

Evandro de Araújo Jardini
Fundação Educacional de Fernandópolis
Av. Teotônio Vilela s/n
Campus Universitário
15600 – Fernadópolis – Brazil
(+5517) 3442 6844
eajardini@yahoo.com.br

Dilvan Moreira
University of São Paulo
SCE-ICMC-USP Caixa Postal: 668
Av. Trabalhador Saocarlense 400 13560-970
São Carlos-Brazil
(+5516) 273 9584
dilvan@icmc.usp.

Abstract

This work presents the Agents 2 system, developed using software agent technology and the Java language, to automate the generation of VLSI leaf cells. It also highlights the interconnection prediction methods used in the system and show how they contribute to help layout generation.

Many agents (implemented as servers) compose the Agents2 system: one Placer server (to place the cell's components) and many Router servers (to interconnect the components in a cell). The servers have the capacity to explore parallel computers (SMP) and to work distributed over a computer network. System scalability increases as new CPUs are added to the computers or new computers are added to the network.

Keywords

Software Agents, VLSI generation, leaf cells, ASIC, Java.

1. Introduction

The project of VLSI (*Very Large Scale Integration*) circuits is usually based on a hierarchy of modules. Modules in the superior levels are composed by submodules, which are formed by other submodules and so on. At the end of the hierarchy, there are modules formed only by transistors. Those modules are denominated leaf cells [1].

The traditional way of creating an ASIC (*Application Specific Integrated Circuit*) layout requires a human planner interacting with a CAD (*Computer Aided Design*) program. It uses a layout methodology based on libraries of leaf-cells. That methodology is good for ASIC projects because the processes performed for layout generation (mainly positioning and routing of components) can be automated, thus reducing the time spent in the project and increasing its reliability [1].

However, that methodology has inconveniences in relation to the maintenance of the leaf-cell library (for instance updating of the manufacture process) and to the availability of specific cells (the variety of existing library cells is limited). As a consequence performance of the designed circuit may be sacrificed [1].

One solution for those problems is the generation of specific leaf-cells as needed by circuits, using tools for automatic generation of leaf-cell layout. The tools should be able to generate layouts for a great number of SSI (*Small Scale Integration*) circuit cells using different manufacturing processes. Since those tools would produce cells that could exactly fit the needs of the project for different process technologies, they would solve both of the main inconveniences mentioned.

The *Agents 2* system [2] has exactly the goal of generating leaf-cell layout for integrated circuits automatically (for different process technologies). This system is an improvement over the *Agents* system [3]. The *Agents 2* system was developed using the Java language and is formed by two programs:

- **The Placer Server:** It positions the components that belong to a circuit and generate several layouts with different placing.
- **The Router Servers:** They wire the trails that interconnect the components positioned by the Placer in a layout.

This new system architecture improves greatly over the old *Agents* design (written in C++ and Lisp). It is a new program using the Java language, making it portable across many computer platforms and operating systems, without the need for code recompiling. The use of the Java language facilities for multithreaded programming makes the new system capable of exploiting parallel processing in Symmetric Multiprocessing (SMP) machines. The C++ language, used for the development of the old Agents, was not capable of easily exploiting parallel processing.

The networking system was improved and the programs simplified (for instance, the Java system now takes care of garbage collecting unused memory) allowing the *Agents 2*

system to work with only two programs: Placers and Router servers (The former system used four).

This article describes the technologies and operation of the *Agents 2* system, how it uses interconnection prediction and shows some results of tests performed generating leaf-cells.

2. The Agents 2 System

The *Agents 2* system is a program created to generate leaf-cells layouts automatically in the CMOS, BICMOS and mixed digital/analogue circuit technologies. A key characteristic added to the version 2 of the system *Agents* is the capacity of the *Router servers* to explore the parallelism provided by multiprocessor computers (SMP), while it performs the wiring of the circuit trails.

The circuit layout generated by *Agents 2* does not need a virtual grid. It is a mask layout ready to be used in an integrated circuit layout. The program does not need virtual grid either during the positioning of the components or during their routing. All operations are executed at mask level.

2.1 Positioning of Components

The fabrication rules that the layout has to follow (which vary according to each production process) are read from a file by the *Placer server* and sent to the *Router servers*, before any routing takes place. The circuits to be generated are received by the *Placer* server in the EDIF format (Electronic Design Interchange Format).

After receiving a circuit, the *Placer server* executes the following tasks:

- Generation of columns of FET (Field Effect Transistor) transistors that have their gates interconnected or groups of other interconnected components (bipolar transistors, for instance).
- Union of the FET columns in order to form groups. To be united, FET columns have to share a certain number of connections through drain or source. The idea is to unite FETs to layout them in the same diffusion line.
- Positioning of this component groups using the genetic algorithm. The resulting placed circuit is then sent to a Router server for routing. Several circuits are placed and sent for routing until one of them is successfully routed.

Since the positioning of components in a circuit is a faster task than routing them, the system uses only one *Placer server* and several *Router servers* working together. Each placed layout is sent to a *Router* that will try to connect the positioned components. If successful the *Router* sends the finished circuit back to the *Placer*.

2.2 Wiring of Circuits

The *Router server* was developed imitating the way human designers use a CAD system (Computer Aided Design) for routing circuits. The designer makes all the important decisions about the project, such as: where the wires will be drawn, the quality of the routing, the need of rewiring, etc. The CAD system supplies the designer with tools to manipulate the project. In the *Router server*, the role of the designer is performed by the *RouterExpert* and *Connect* software agents and the role of the CAD system is performed by a *Design* object.

This *Design* object keeps the data of the project as well as the methods to analyze or change it. The *Design* object provides the necessary ways to collect information about the project (if two wires are colliding, etc.) and the methods to implement changes in the project (to insert or destroy wires, etc.). The *Design* object's resources are used by the agents *RouterExpert* and *Connect*.

2.3 RouterExpert and Connect Agents

The *Router server* was developed using software agent technology. Agents are software components that communicate with their pairs through messages using a communication language [4]. That characteristic allows agents to cooperate among themselves. They can be as simple as a subroutine or bigger entities such as autonomous programs [1]. The software agent technology is used at two levels:

- One *RouterExpert* agent is in charge of creating and eliminating *Connects* agents, verifying the quality of the routing, determining the wires cost, etc.
- Many *Connect* agents are in charge of exploring and finding interconnection routes among the circuit components.

The behavior of each *Connect* agent is simple. But the idea is to have several agents working together and in a parallel way (instead of having one complex program full of production rules). The more *Connect* agents can be executed in parallel, the less time it will take to route a circuit.

The agents *RouterExpert* and *Connect* execute the routing by using a variation of the maze routing algorithm. The maze routing algorithm is based on a rectangular grid that represents the circuit. The circuit is mapped on the grid and it is determined which cells are free or blocked. To each cell a cost is attributed, forming a structure similar to a search tree. Then an algorithm is used to analyze and expand the vertexes of the tree generated to find the objective.

A variation of that algorithm introduces the concept of interesting points [5]. Instead of expanding the vertexes to all the directions, *Connect* agents expand them directly to

an interesting point. A point is considered interesting when it is aligned with the objective point or with the extremities of an obstruction. Actually, to the extremity of an obstruction is added a security margin proportional to the minimum distance from the obstruction to the wire being routing plus half of the wire width.

2.4 *Router* Agent Operation

The *Router server* receives the circuit serialized from the *Placer server* and instantiates a *Design* object to contain this circuit. Starting from this point, the *RouterExpert*

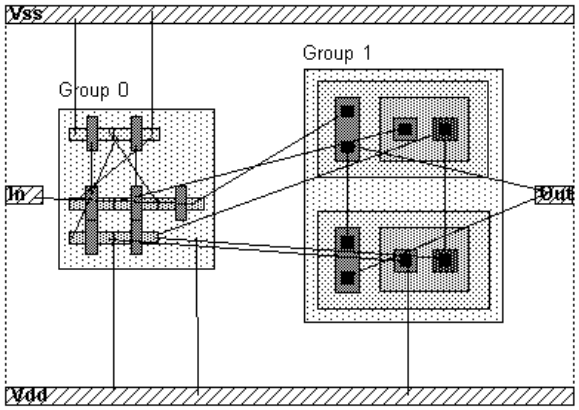

Figure 1: Rat's nest" fashion.

and *Connect* agents will use this object to perform the circuit routing. First the *RouterExpert* will do the simple routings, connecting the points that are aligned in polisilicon or diffusion.

To connect the other components, the remaining circuit nodes are put in a list, ordered by importance and size (the smallest ones first). From that point on, only the polisilicon, metal1 and metal2 layers will be used for connection (the program currently works with only two metal layers).

In the *Design* object, each node contains a list of subnets that partially connect the node components. A subnet is the group of wires that interconnect part of the components that belong to the same circuit node. The method *connectNode* of the *RouterExpert* agent uses this list to connect the subnets. The method *connectSubnet* does this job. If the method does not manage to connect a subnet, it puts it at the end of the list (to give it a second chance later) and tries to wire the next one. If at the end, the method does not manage to connect all subnets, an error condition occurs and this placed circuit is considered not wirable.

The *subnet* connections are based on the interesting points and are performed by the *Connect* agents. They analyze the interesting points that are closer to them running in parallel as Java execution threads (one agent per thread, a Java thread is computationally inexpensive). Starting from this analysis, several operations can be performed by *Connect* agents: they can connect the wire with its destination (and offer it to the *RouterExpert* as a possible best wire), create a new wire section or replicate. When they replicate, a new *Connect* agent is created for each new interesting point. The agents will reproduce until they find their objective or run out of interesting points. In the

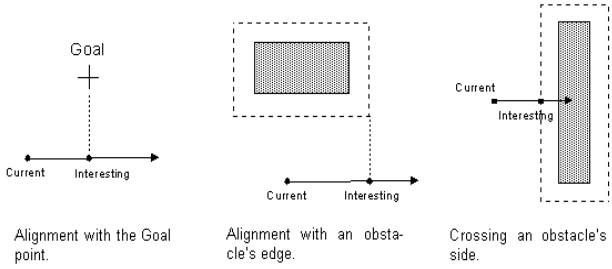

Figure 2. : Interesting Points.

wiring process an agent is always concern with the cost of its wire. If the cost of its wire becomes larger than an already obtained by the *RouterExpert*, the agent dies. This avoids unnecessary processing and allows other agents to assume the control of the processors.

The *Connect* agent's goal is to reach a point in another subnet in the same node to join the two. In this process, they should join all subnets belonging to a node. This node is them considered wired. If all nodes of a placed circuit are wired, the routed circuit is sent back to the *Placer* agent.

3. Interconnection Prediction

After showing how the *Agents 2* system work, it is useful to take a closer look in how it predicts wire interconnections. This has to be done at two levels:

- By the *Placer server* during evaluation of placements generated by the genetic algorithm.
- By the *Router server* during actual wiring.

3.1 Estimation During Placements Selection

The *Placer server* uses the genetic algorithm to generate new circuit placements. The new placements should be evaluated to find how good they are. This step is more closely related to the system being optimized than to the genetic algorithm itself. The best way of doing the evaluation would be by actually routing each placement using a *Router server*. Unfortunately the routing process is slow and would take too long to evaluate all candidates. In place of a full routing, another method is used to estimate the cost of wiring the circuit. The goal of this method is to evaluate how much a particular placement

may contribute to the total length of wire and to the amount of crossings among the wires of the circuit. Figure 1 shows the wire connection as straight lines in a "rat's nest" fashion, from that projection one can see how much this particular placement will influence the routing process.

To get a similar effect, the evaluation routine uses the same wiring algorithms that the *Router server* uses, but it allows crossing over and short circuits to take place and it does not test for design rule violations. All connections are carried out with the smallest wire possible, changes of layer, however, are undertaken whenever necessary.

As this evaluation pseudo routing takes place, the cost of each wires is computed. The routine that calculates the cost of each wire is the same that is used by the *Router server*. The number of crossings of each new wire with all the old ones, already laid out, is accumulated. At the end, the evaluation routine has the total cost of the pseudo routing and the total number of crossings. Each placement will be judged by these two values.

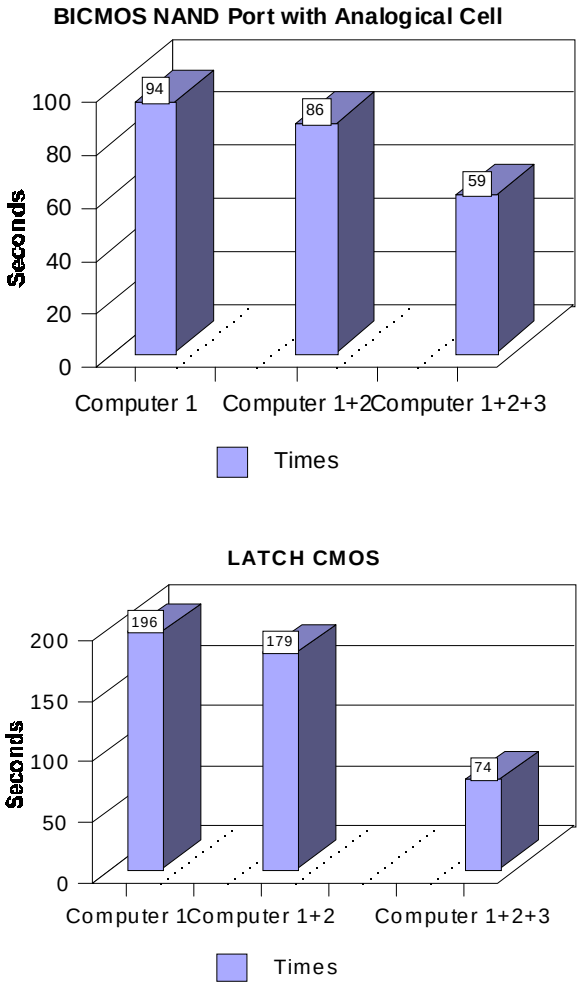

Figure 3. : Results of the times in a distributed environment.

This interconnection evaluation method is somewhat coarse, but it is appropriate for this level of analysis. The genetic algorithm generates thousands of placements; it would be impossible (with the available technology) to wire them all. However, it should be pointed out that hundreds of placements will survive this early pruning. The idea is to throw out just the very bad ones. Later, the surviving placements will be properly wired before a final circuit is found. There are many distributed *Router servers* to do this job.

3.2 Estimation During Actual Wiring

The *Router server* uses a form of maze routing based on interesting points. Maze routing can be performed over a rectangular grid of cells, with some cells free and others blocked. Basically, the algorithm finds the shortest path between two predefined cells that does not pass through any blocked cell.

A performance optimization, which can be employed in the maze routing algorithm, is expansion directly to the next *interesting point*, where a change in direction or layer is more likely [5]. A point is defined as *interesting* when it aligns with the goal point, obstacles' edges or crosses obstacles' sides (Figure 2).

This estimation saves time skipping over less interesting parts of the layout and, more importantly, by eliminating the need to process data at the costly level of pixels, processing is performed directly on the circuit description.

4. Results

The motivation of this work was to help the development of better tools for circuit mask generation using the inherent parallelism of distributed systems and multiprocessor SMP computers. The idea is that, as the program uses these resources, the time spent in the process of circuit cell generation decreases.

The first goal of the tests presented here is to show that the extra routing work, generated by the use of a coarser interconnection estimation method (in the Placer router), can be handled by the distributed Router servers. In other words, that the routing is scalable with the number (and power) of available computers.

The following machines were available to perform the tests:

- Computer 1 - Intergraph computer with 4 Pentium Pro processors of 200 MHz - Windows NT 4.0.
- Computer 2 - Computer with 1 Pentium II processor of 266 MHz - Windows NT 4.0.

- Computer 3 - Computer with 2 Pentium II processors of 350 MHz - Windows NT 4.0.

The graphics of figure 3 show the obtained results. They demonstrate that the *Agents 2* system is able to scale up in distributed environments and, in doing so, to handle the extra routing load.

The results of the performed tests can vary a little from one program execution to another due to factors linked to computer configuration differences and the workload of the network. Unfortunately it was not possible to test the system in other architectures (Sparc, PowerPc, Alpha, etc), as the machines to do so were not available.

The second goal of the tests was to show that the Agents 2 system (using the two interconnection estimation methods) is able to generate reasonably good layout for leaf cells. Figure 4 is one example of such layouts.

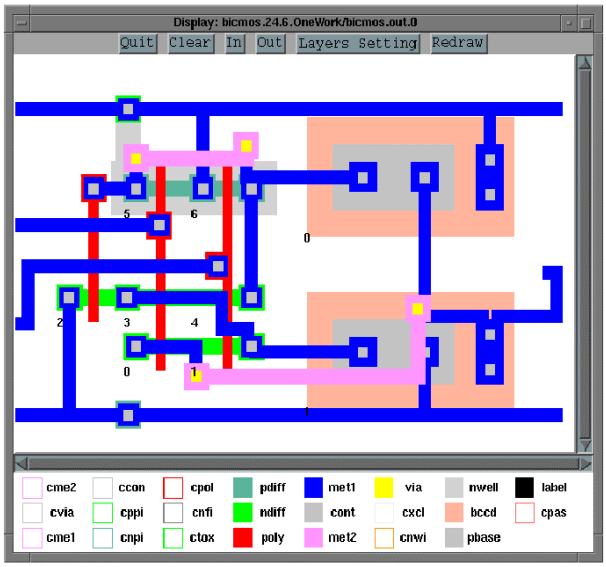

Figure 4. : Completed routed circuit.

5. Conclusion

This article presents the *Agents2* system, a tool for automatic generation of leaf-cell mask layout. The system has two parts, the *Placer* and *Router* servers.

Besides generating mask layout for a variety of technologies, the system is scalable in distributed environments (such as local networks or computer clusters).

The system technology seems very appropriate for pure analog cell generation, but no work has been done to test this idea. The *Agents2* system is freely available for download (under the GNU Public License) at the site http://java.icmc.usp.br/research.

The system is still a prototype, and many improvements have to be made to integrate the system in commercial layout generation tools.

6. Acknowledgments

The authors gratefully acknowledge the São Paulo State Research Foundation (FAPESP) for financially supporting this work.

7. References

[1] Moreira, D.A. and Walczowski, L.T., “Using Software Agents to Generate VLSI Layouts,” *IEEE Expert Intelligent Systems*, Vol. 12, No. 6, November/December 1997, pp. 26-32.

[2] Jardini E.A. and Moreira, D.A, "Designing VLSI Circuit Masks with the Software Agents2," *Proc. of XIV Symposium on Integrated Circuits and Systems Design*, Pirenópolis Brazil, 2001, ACM, IFIP WG10.5, IEEE Computer Press, pp. 168-173.

[3] Moreira. D.A. and Walczowski, L.T., "The Development of Very Large Scale Integration Systems", *Knowledge Based Systems Techniques and Applications*, Vol IV, Chapter 37, pp. 1227 - 1271, Academic Press, January 2000, ISBN 012443875X.

[4] Genesereth, M.R. and Ketchpel, S.P., “Software Agents,” *Communications of the ACM*, Vol. 37, n.7, July 1994, pp. 48-53, 147.

[5] Arnold, M.H. and Scott, W.S., “An Interactive Maze Router with Hints,” *Proc. Of the 25th ACM/IEEE Design Automation Conference*, 1988 IEEE, pp. 672-676.