AGENTS: A Distributed Client-Server System for Leaf Cell Generation

DILVAN de ABREU MOREIRA
Department of Semiconductors, State University of Campinas, Brazil
LES T. WALCZOWSKI
Electronic Engineering Laboratory, University of Kent, U.K.

The AGENTS system is a set of programs designed to generate automatically the mask-level layout of full custom CMOS, BICMOS, and bipolar leaf cells. The system is formed from four server programs: the Placer, Router, Database, and Broker.

The Placer places components in a cell, the Router wires the circuits sent to it, the Database stores all the information that is dependent upon the fabrication process, such as the design rules, and the Broker makes the services of the other servers available.

These servers communicate over a computer network using the TCP/IP Internet Protocol. The Placer server receives from its client the description and netlist of the circuit to be generated using EDIF (Electronic Design Interchange Format). The output to its client is the mask layout of the circuit, again codified in EDIF. The concept of agents as software components which have the ability to communicate and cooperate with each other is at the heart of the AGENTS system. This concept is not only used at the higher level, for the four servers, but at a lower level as well, inside the Router and Placer servers, where small relatively simple agents work together to accomplish complex tasks. These small agents are responsible for all the reasoning carried out by the two servers, as they hold the basic inference routines and the knowledge needed by the servers. The system's philosophy is that competence should emerge out of the collective behavior of a large number of relatively simple agents. In addition and integrated to these small agents, the system uses a genetic algorithm to improve components' placement before routing.

Categories and Subject Descriptors: B.7 [**Hardware**]: Integrated Circuits; B.7.2 [**Integrated Circuits**]: Design Aids—*placement and routing*

General Terms: Design

Additional Key Words and Phrases: Client/server model, genetic algorithms, software agents

This work was sponsored by CNPq—National Council for Research, Brasilia DF, Brazil.
Authors' current addresses: D. de Abreu Moreira, Dept. of Semiconductors, Instrumentation and Photonics, Electrical Engineering Faculty, State University of Campinas, Brazil; L. T. Walczowski, Electronic Engineering Laboratory, The University, Canterbury, Kent, CT2 7NT, U.K. Tel: +44 1227 823713, Fax: +44 1227 456084; email: L.T.Walczowski@ukc.ac.uk.

1. INTRODUCTION

The traditional way of creating the layout of ASIC (Application Specific Integrated Circuit) custom chips requires a human designer to interact with a CAD (Computer Aided Design) program. ASIC design is normally based on a hierarchical specification structure. High-level modules are composed of submodules, which are formed by smaller sub-modules, and so on. At the end of this tree-like structure are modules formed only with transistors. These modules are called *leaf cells*.

A major drawback of this methodology lies in the design and maintenance of the leaf cell libraries for every upgrade of the manufacturing processes. Additionally, as the number and the variations of cells are both limited, some required cells may not exist in the libraries. Circuit performance will then have to be sacrificed.

The AGENTS system is a set of programs designed to generate automatically full custom CMOS, BICMOS, and bipolar leaf cell layout. Additionally the system can handle analogue cells inside digital designs. The system consists of four server programs: the Placer, Router, Database, and Broker. The Placer places components in a cell and uses the Router to wire them; the Router wires the circuits sent to it; the Database keeps all the information that is dependent upon the fabrication process, such as the design rules, and the Broker makes the services of the other servers available and manages the available resources. The four servers can run in parallel distributed through out different machines to solve cooperatively a placement-routing problem.

The Placer server receives from its client, via the network, the description and netlist of the circuit to be generated using EDIF [1984] (Electronic Design Interchange Format). EDIF is a standard Lisp-like language used to represent electronic designs. The output to its client is a design rule that is a correct mask-level layout of the circuit, again codified in EDIF. The distributed architecture of AGENTS takes advantage of the current trend in mainstream computer hardware and software towards distributed parallel solutions. The system uses techniques that allow electronic design automation (EDA) tools to exploit now the advantages of what will be mainstream computing.

2. PREVIOUS WORK

Much work has been undertaken to automate layout generation. This work can be divided into two groups: knowledge-based systems (basically rule-based systems) and systems based on intensive search algorithms, such as Lee's algorithm, simulated annealing, or simulated evolution. Design systems in the first group use mainly heuristic expert knowledge, in the form of production rules, to guide their search in finding solutions for the layout problem. For example:

—Talib [Kim and McDermott 1986] is a rule-based NMOS module compiler with more than 2100 rules. It treats algorithmic-based procedures as

subtasks while supervising them with a knowledge-based control system. Talib relies on its control knowledge to decide when and how to perform a specific subtask properly. Talib demonstrates how to use clusters of circuits with known layouts to complete an NMOS cell layout, and how to take input boundary conditions into account. Most of Talib's design knowledge is based on empirical rules used by human experts when working in design examples. Talib is able to generate a compact layout for small circuits, and its rule-based approach makes it easy to add new knowledge.

—Topologizer [Kollaritsch and Weste 1985] uses rules specified by an expert designer to produce a symbolic CMOS layout from the descriptions of the transistor connections and the environment in which the cell resides. The placement rules include moving transistors between locations, exchanging their positions, and rotating them. The routing expert consists of a prerouter and a refinement router. The prerouter produces "rough routing" by assigning a unique track to each pair of terminals to be connected. The refinement router then improves the routing by applying a set of rules to eliminate bad routing. By using MULGA, a symbolic layout system, outputs can be translated into a mask layout.

—LES (Layout Expert System) [Lin and Gajski 1988] is a random logic module layout generator, targeted for single-metal polysilicon gate CMOS technology, in hierarchical VLSI designs. It applies rules and algorithms based on a multirow layout style. LES takes a top-down strategy that generates leaf cells after, rather than before, placement and global routing are undertaken. No detailed routing is needed because the cells are laid to fit their environment. LES consists of seven expert systems organized in a blackboard architecture: analysis, architecture, placement, characterization, layout, evaluation, and optimization experts.

Design systems in the second group use mainly algorithmic techniques, such as Lee's algorithm, simulated annealing, or simulated evolution, to guide their search in finding solutions for the layout problem. For example:

—Excellerator [Poirier 1989] is a program that generates CMOS cell layout from circuit specifications. The program generates CMOS layout that is "gate-matrix-like," but not constrained by strict "gate-matrix" design rules. It supports different layout shapes and port location constraints. Multirow transistor placement is undertaken by identifying groups of serially connected transistors and then positioning and ordering the groups. A routing technique based on a recursive version of the A-star search algorithm is used. Routing priority can be given to critical nets.

—Lib [Hsieh et al. 1991] uses a branch-and-bound search strategy to find an optimal chaining of transistors. It folds large transistors into multiple columns to meet the cell height constraint. The whole cell is divided into five routing regions: Two regions are in the diffusion islands, and the other three are rectilinearly shaped routing channels (two between the

power rails and diffusion islands and one between the PMOS and NMOS diffusion rows). The program uses a graph theoretical method to select subnets for routing in the diffusion island. A global routing algorithm assign the remaining nets to the three rectilinear channels. Detailed routing is done using SILK, a router system.

—PROCORE (PRObabilistic COnflict REsolver) [Mossa et al. 1994] is a system based on simulated annealing optimization techniques. Simulated annealing is used to provide a framework for controlling ripup and reroute transformations within an implementation of Lee's algorithm. Intersections between nets are removed individually by rerouting one net (selected randomly) involved in an intersection, such that it does not cross the other intersecting net (crossing other nets is still permitted). The resulting transformation is either accepted or rejected based on the evaluation of a global cost function.

AGENTS differs from previous systems in that it emphasizes flexible design and the use of new ideas (e.g., the concept of agents as software components). Flexibility is achieved at design level using object-oriented programming [Moreira and Walczowski 1995], using scalability at performance level, and by being a portable and easy to integrate system at integration level. More important, flexibility is delivered to the user as a richer set of layout options; the AGENTS system can take BICMOS and CMOS technologies and handle small analogue cells within a digital design.

Scalability means that a program should take advantage of the resources of the hardware it is running on to improve its performance, be it a PC or a network of powerful workstations. In the AGENTS system, scalability is achieved mainly by the use of the client-server model.

3. SCALABILITY AND SOFTWARE AGENTS

Fully scalable system architectures are important in today's computer industry; that is, when the number of processors in a system is increased, system performance should scale up proportionally. For software, scalability means that a program should adapt to take advantage of the available computational resources.

Scalability is brought into the AGENTS system by software agents—software components that communicate with their peers by exchanging messages in a communication language [Moreira and Walczowski 1995]. Using this concept, systems can be divided into smaller entities that can work independently, or together, to solve problems cooperatively. While agents can be as simple as subroutines, they are usually bigger entities with some sort of persistent control and autonomy. Their main characteristics include the ability to communicate and cooperate with others.

The agent metaphor is employed at two levels: at the system level, where the four servers that form the system can run in parallel on different machines, and at a program level, where *agent objects* are used to control and coordinate routing and placement tasks inside the program.

Agent objects are objects with an expert system embedded in them. These expert systems use a simplification of the problem-space computational model proposed by Newell [1990]. They create *problem spaces* to search for a solution. Inside these spaces they have *states* and they apply *operators* to find new states during the search process. *Agent objects* perform searches until they reach their *goal*.

Agent objects generally work as a group. Sometimes different kinds of agents work in the same group and have their own "personality" and aim in life. Personality is determined by the set of behaviors the agents can perform, similar to insect-like autonomous robots [Brooks and Flynn 1989]. Changes in behavior can be dictated by an agent's perception of changes in its environment or they can be directly commanded by another agent.

In contrast to the more traditional solution where complex behavior comes from one big monolithic expert system with thousands of rules, here it comes from the interaction of many simple expert systems with a small number of rules.

3.1 Architecture of Multiagent Systems

Software agents can be used to partition large software into smaller units. Programs can communicate and use each others resources to accomplish a bigger task. Because they still are independent, they can run in parallel on different machines, thus improving performance.

The AGENTS system uses four *agent servers* that can run in parallel on different machines. To organize all the agents of the system, a solution based on a broker was adopted. A broker is one who acts as an agent in negotiating contracts. In terms of distributed computing, the broker provides an intermediary between the client making a request and the server that fulfills the request.

An *agent server* called Broker was created to coordinate the access of applications to the placer, router, and database servers. It was written in Lisp (Squeme), and it can interpret Lisp commands sent in by a client. On top of Lisp, the Broker implements a subset of the KQML (Knowledge Query and Manipulation Language) [Finin and Weber 1994]. KQML is a language for programs to communicate attitudes about information, such as querying, stating, believing, requiring, and subscribing.

The Broker agent server retains information about its clients and about the other three servers. This information forms the Broker's virtual knowledge base (VKB). Each client or server is identified by a symbol (generally a number) and has an entry in the VKB. Like the other three servers, the Broker answers its questions in EDIF.

The Broker performs a lot of the housekeeping that would otherwise have to be undertaken by its clients. It will start the Database agent server. It will start a requested server, Placer or Router, for a client if they are not available. It can start the servers on predefined hosts. The client does not have to worry about the network address of any server. The Broker decouples the clients from the implementation details of the servers.

The kind of placement and routing undertaken by the Agents system is process independent. To achieve this independence, all the information about the process rules is kept in a process Database server separate from the Placer and Router servers. Whenever any server needs information about the process, it queries the database.

The Database server stores its information in hash tables, the information is accessed using a very simple language. The queries state the type of the information they are looking for and to which elements (usually layers) this information refers. For instance, the query:

```
(minSpacing ndiff cont)
```

looks for information concerning the minimum space between two elements: the layers Ndiff and Pdiff. The answer for all queries is sent back in EDIF. For example, the answer to the minSpacing query is (E 15 −7), which represents 1.5 μm. The data for each particular process is read from a process description file.

4. PLACEMENT

The Placer *agent server* undertakes the placement of circuit cells in a defined area. After reading the design information from its client in EDIF format, the server performs the placement of components in three steps:

(1) It forms columns of related components.
(2) It forms groups joining columns of fets that share drain or source connections or form pass pairs.
(3) It places the groups using genetic algorithm techniques and uses the router server to route them.

The best placement successfully routed becomes the final circuit. This circuit is then sent back to the client application. The design is formed by a list of EDIF cells containing physical mask layout views and one cell, the main cell, containing a symbolic view reporting how the other cells are to be connected.

Apart from the main cell, there are five types of cells: Pads, for I/O pads; Fets, for MOS transistors; Bipolar, for bipolar transistors; General, for any other kind of cell; and ElectricNode, for the wiring of the nodes. The server places component cells over a design containing the pad cells and any preplaced cells. The server can accept partially placed or wired layouts for completion. Figure 1 shows a possible input design and the three types of cells to be placed: CMOS fets, bipolar transistors, and general cells.

4.1 Column Formation

There are three kinds of *agent objects* for placement: the *Cont* agent controls all the operations; the *Abutted* agent first builds columns of related transistors and subsequently forms groups, and the *Eval* agent uses the genetic algorithm to find a good placement for the groups. The Cont agent coordinates all the actions. It receives the new circuit as a list. It separates

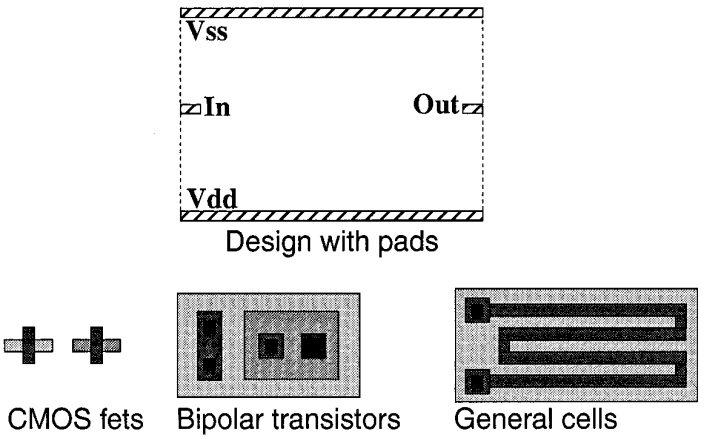


Fig. 1. Types of cells used by the Placer server.

the NMOS fets, PMOS fets, bipolar transistors, and general cells in different lists and creates the first Abutted agent.

An Abutted agent has two behaviors. When it is created it performs its first behavior: It goes to the lists of available cells, in the Cont agent, and tries to grab as many cells as it can to form the biggest possible column following the rules.

—While there are MOS transistors available, it tries to obtain MOS fets that have their gate interconnected or form a pass pair. A pass pair is two fets that have their source and drain interconnected.

—If all MOS transistors are gone, the Abutted agent tries to match pairs of interconnected bipolar transistors. If it can not form a pair, it settles for just one transistor.

—If all the transistors are already gone, the Abutted agent grabs a general cell. There can be only one per agent.

After an Abutted agent has formed its column, it tries to reproduce. If there are more cells to obtain in the Cont agent, a new Abutted agent is created and tries to obtain its own set of cells. Abutted agents behave like a culture of bacteria—they keep reproducing until there is no more food.

4.2 Group Formation

When the column creation process finishes, the Cont agent switches the Abutted agents to their second behavior. The Abutted agents will now try to pair up with other agents joining their cells. To pair up, two agents have to share a number of source/drain interconnections. The idea is to join columns of fets, by either the source or the drain sides. In this way fets can be laid out in parallel strips of diffusion; they can be abutted.

The group formation process is performed in cycles controlled by the Cont agent. In each cycle, the Cont agent goes through the list of Abutted agents, exposes each agent to the others, and asks the other agents how well they

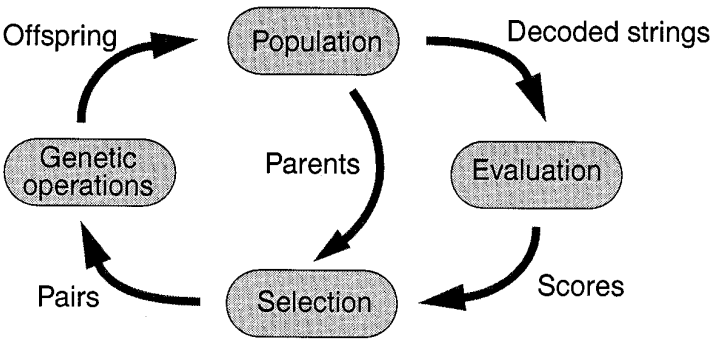


Fig. 2. The "reproduction" cycle.

connect to this agent. Each agent then creates a report showing its situation relating to the exposed agent—stating whether a match is possible, how good this match is, and details how it should be implemented. At the end of a cycle, the two agents that have the best connections between each other are joined. These cycles continue until no agents can be joined or the quality of the possible connections is too poor.

After the grouping of cells has finished, the Cont agent contains a list of Abutted agent holding groups of cells. It then takes these cells from the agents and puts them in a list. The Abutted agents are then destroyed.

4.3 The Genetic Algorithm Placement

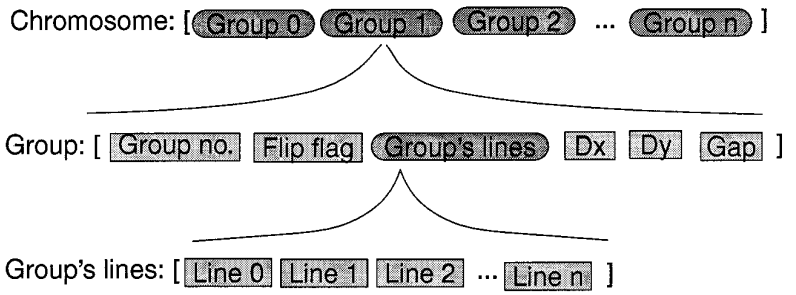
Up to now, only topological relationships between the cells of the design have been exploited to create the groups of cells held by the Cont agent, no grid or position coordinates have been determined for them. This group will now be placed using the genetic algorithm encapsulated in the Eval *agent object*. Genetic algorithms are a class of computational model that mimic natural evolution to solve problems in a wide variety of domains [Filho et al. 1994]. Even though *agent objects* use knowledge to reduce search time, a great deal of searching is still necessary.

The algorithm operates through a simple cycle: creation of a population, evaluation of each individual, selection of the best ones for reproduction, and genetic manipulation to create a new population (Fig. 2). The cycle is repeated until a good result is found. The inputs for the Eval agent are an empty design and a list holding the groups of cells to be placed.

Initially, the agent randomly generates some individuals. These individuals are then evaluated and classified and become the initial population and the cycle (shown in Figure 2) can begin. There are four main steps in this cycle:

Population. Each individual in the population is encoded into a representation, its chromosome, to be manipulated by the genetic operators (crossover and mutation). Figure 3 shows an individual and the chromosome that represents it. The chromosome is represented by a list. As the top of Figure 3 shows, the elements of this list represent each group being laid out in the

Coding structure:



Example:

Chromosome: [0 FLIP [1 0 2] 10 12 5] [1 NO_FLIP [1 0] 8 5 3]

Actual individual:

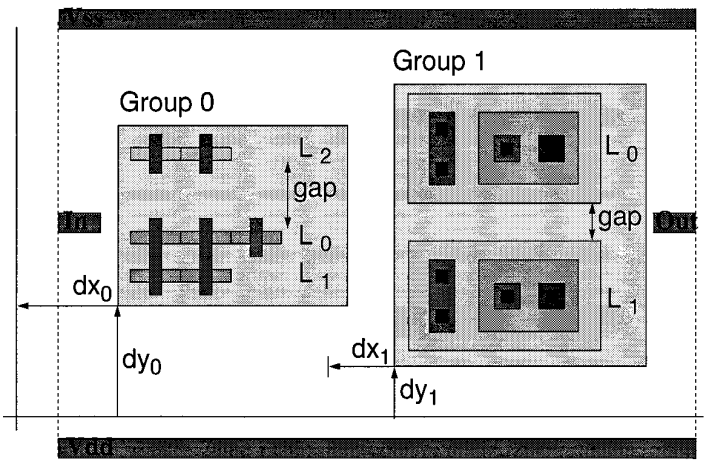


Fig. 3. Coding schema.

cell, with the groups laid out in the same order as they appear in the list. For each group there is another list describing how that particular group will be laid out. With the information provided by the chromosome, the individual, shown in Figure 3, is laid out.

Evaluation. The best way to do the evaluation is to actually route each individual using the router server. Unfortunately the routing process is slow, and would take too long to evaluate all candidates. In place of full routing, a method to estimate the cost of wiring the circuit is used. The evaluation routine uses the same wiring algorithms that the router uses, but it allows crossing over and short circuits to take place, and does not test for design rule violations. All connections are carried out with the smallest wire possible. The cost of the wires and the number of wire crossings are computed. Each placement is judged by these two values.

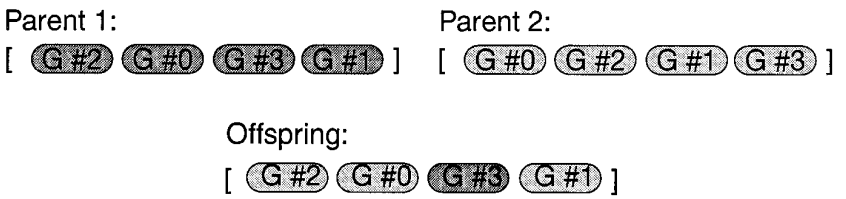


Fig. 4. Crossover in lists.

Selection. The whole population of individuals is kept in a list ordered by fitness, the fittest individuals coming first in the list. Individuals with smaller wire costs and crossings are considered fitter. Every new individual has to be appended to this list in the appropriate position. As the position of an individual in the population list increases, its probability of being selected for reproduction decreases, and its probability of dying increases. When the population reaches a predetermined maximum number, half of it is killed.

Genetic operations. As Figure 4 shows, when two individuals mate, their genetic materials mix, in this case the list containing placement information for each group. In Figure 4, parent 1 was randomly chosen as the main parent, the order of groups in the offspring's chromosomes reassembles, but the actual group's placement information came randomly from both parents. After an offspring is generated, swap and mutation operations can be applied. At the top level, mutation can be used to swap the position of some groups in the offspring. Inside each group description, the FLIP flag can be changed by mutation or the group's lines can be swapped.

The reproduction cycle is repeated until a certain number of generations has been tried. The best placement is then sent to a Router server for wiring, and the reproduction cycle is restarted (for another number of generations) to produce another best placement. The Placer keeps doing that until one of the Router servers produces a fully routed design (or an error condition occurs), this design is then sent back to the Placer's client.

5. ROUTING

The Router *agent server* performs all the circuit wiring in the AGENTS system. It receives circuits in EDIF format with all components already placed. These circuits are then routed and returned to the client.

The Router server mimics the way human designers use a CAD (Computer-Aided Design) system to route circuits. The designer makes all the important decisions about design, such as where the wires are going and about the quality of the routing. The CAD offers the designer a tool to represent and manipulate the design. The designer is in charge of the decision making process and the CAD system offers the medium and facilities to implement his or her decisions. In the Router server, the designer's role is carried out by the RouterExpert *agent object* and the other

agents under its supervision, and the CAD role is carried out by the Design objects.

5.1 The CAD Role

The Design object holds the design medium and the methods to analyze or change it. The Design object has two groups of methods to perform its functions: *building* methods, for changing the design data, and *retrieve* methods, for collecting information.

Retrieve methods search for information about the components of the design. The search can be based on a component's individual characteristic, on spatial relationships between components, or spatial constraints—such as distance. These methods can return Boolean statements, spatial information, or a list with references to components.

The two most important spatial relationships are *crash* and *touch*. A crash happens when any section of a new wire breaks any design rule in relation to any section belonging to any wire already in the design. A touch happens when a proposed wire section touches any section belonging to any wire in a list, both sections should be in a wiring layer. Both relations can be tested for individual wire sections or for pointers representing a wire that spreads in a defined direction and has infinite length. In the latter case, a list is returned containing the component and the distance at which it crashed or touched the pointer.

The Design object's second function, to implement the designer decisions on the design, is performed by the building methods. This group of methods adds elements to the design, and can conduct construction tasks that are not complex enough to demand the help of an expert system.

5.2 The Designer Role

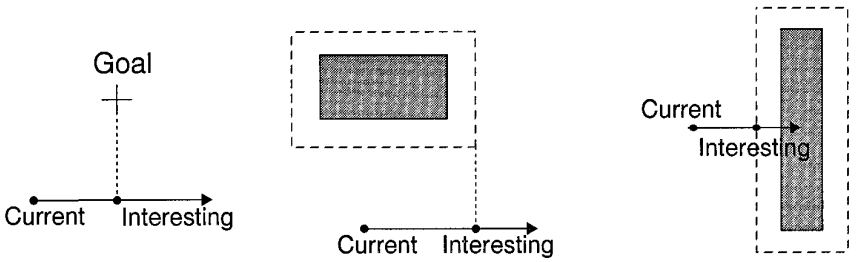
The Designer role is carried out by the RouterExpert and Connect *agent objects*. They are in charge of the decision making process. Their function is to control the way the routing is undertaken using the facilities provided by the Design object.

In combination, these two kinds of agents perform a kind of augmented maze route algorithm. Basically, the agents find a short path between two nets probing the design's *interesting points* [Arnold and Scott 1988] (points where a change in direction or layer is more likely). This saves time, since it skips over less interesting parts of the layout and, more importantly, eliminates the need to process data at the costly grid level—processing is performed directly on the circuit description held by a design object.

A point is defined as interesting when it aligns with the goal point, obstacles' edges, or crosses obstacles' sides (Figure 5). In the case of obstacles, the edge and sides considered are the ones that overlap the obstacle by a security margin.

5.3 Router Expert

When the RouterExpert agent receives a circuit, it first performs some simple routing, such as straight diffusion and polysilicon lines. After the



Alignment with the Goal point.

Alignment with an obstacle's edge.

Crossing an obstacle's side.

Fig. 5. *Interesting* points.

basic connections have been completed, the RouterExpert agent wires all remaining connections. The unwired nodes are put in a list, ordered by importance and size. This will help if any rerouting has to take place later. The RouterExpert then connects all members of this list. From this stage on, only the wiring layers, in this case Polysilicon, Metal1 and Metal2, will be used for interconnections.

In a Design object, each node has a list of partially routed subnets. Each list contains at least one wire corresponding to a component terminal. All subnets in this list have to be connected. The RouterExpert fails if all subnets cannot be connected together.

The wiring of a connection is undertaken by the connect agents by analyzing the nearest *interesting* points, beginning at the wire origin. A Connect agent is created for each interesting point. From its analysis of the point, a number of operations can be performed: It can change layers, connect the wire to the destination, create a new piece of wire, unwire a blocked path, etc. The agent's goal is to reach a point in the destination subnet (Figure 6). It can create other agents (reproducing) to analyze other nearby points of interest. These operations continue until two subnets are finally wired together by a reasonably short wire.

The RouterExpert agent controls the population of Connect agents and the way they perform the routing. If an individual finds a new interesting point, it reproduces. If it completes a wire, it sends it to the RouterExpert agent. If it has exhausted all its options, it dies. The RouterExpert then takes care of this "farm," killing individuals with costly routing and giving more resources to individuals with prospective wires. Figure 7 shows the data structure inside a RouterExpert. Ideally, Connect agents should have been implemented as parallel routines. Unfortunately this was not supported on the system on which the program was developed. For this reason, the RouterExpert implements a small timesharing machine to run the Connect agents (Fig. 7). When a Connect agent runs and creates new ones, they are ordered in queues. Subsequently the RouterExpert uses various strategies to determine which should run first and which should be killed (for example, because their wires are already too large).

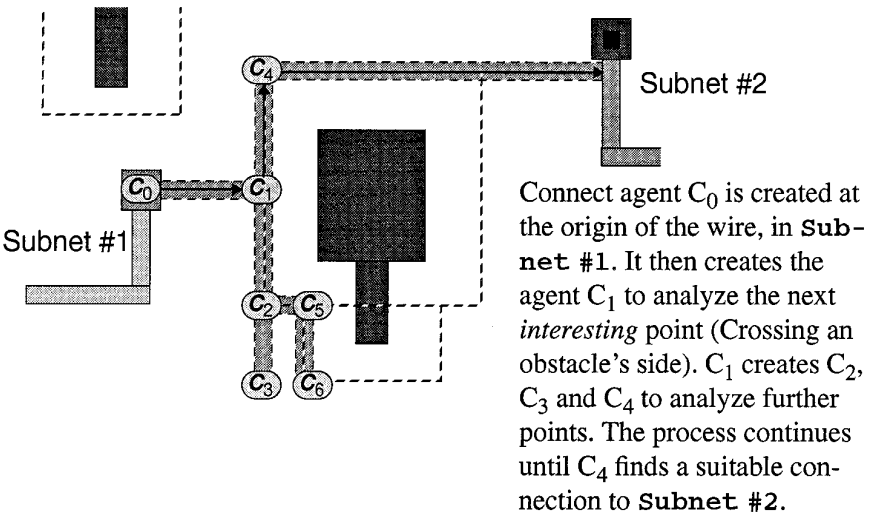


Fig. 6. Connect agents probing *interesting* points.

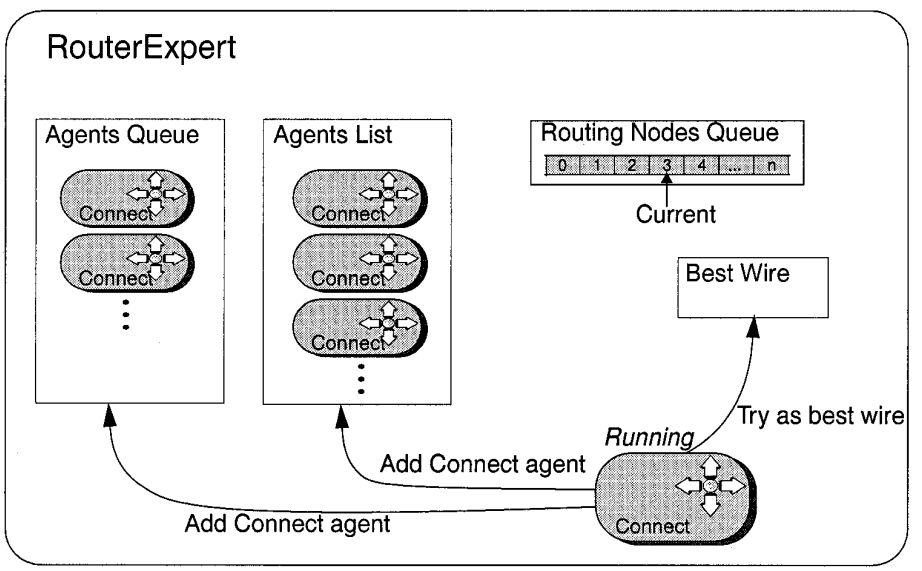


Fig. 7. RouterExpert agent data structure.

When the path of a wire is totally blocked by an already routed wire, the program unwires the old wire to allow the wiring of the new. Subsequently, the old wire is rewired. Unwiring is a very expensive operation. The unwiring of a section of wire can lead to the complete unwiring of many nodes. All this care has to be taken because the wiring of a node can strongly influence the routing of the other nodes connected after it.

5.4 Connection Object Agent

The Connection agent is at the core of the routing process, it carries out the actual routing of the subnets. It is created to analyze an interesting point. When running, it finds other interesting points, extends its wire to reach them, and creates other Connection agents to analyze them. If it finds a connection with the target subnet, it extends its wire to it and proposes it as the best wire to the RouterExpert agent.

When a Connection agent is created it receives data from its parent about its "mission." This data include where the Connection agent is in the design, where it should go, and the wire segment it already holds. The agent then remains dormant until the RouterExpert runs it.

When the Connection agent begins to run, it first checks out its environment and finds out in what direction the final target point for its wire is. With this information it plans which of its four operators will be activated. It feeds them into the Options list, a list that holds operators to be applied, and applies the operators in the list until it is emptied. The applied operators can trigger other operators to be applied (adding them to the Options list). When the agent has tried all possibilities it halts and "dies."

The Connect agent uses four operators to probe its design options. The most important is the Change Direction operator, which is the only one to undertake the final connection to the target net. The other three are the Change Layer (it changes the wire layer to the one specified); Go to XY (it tries to extend the current wire for a specified distance in a specified direction); and Get Round (it tries to go round a specified obstacle in a specified direction).

If successful, all three operators add a new section to their wires and create a new Connect agent to work on the end of this new section. When they create new Connect agents, they send them to the RouterExpert.

The Change Direction operator probes the space in defined directions, looking for a connection in the target net or for obstructions. It gathers information that can be used to activate other operators. Depending on its environment and probe direction, the operator will take some of the actions (A to F) shown in Figure 8.

When the path is unobstructed, a connection is made (action C) and the resulting wire is sent to the RouterExpert agent as a possible best wire. If there is an obstruction, in addition to the actions (A–F), the operator can create a new Connect agent that asks for the obstructions to be unwired and sends it to the RouterExpert agent. If this new Connect agent ever runs, it will unwire the obstruction and try a connection. The RouterExpert will first run all Connect agents that do not require unwiring of nodes; the other Connect agents will be tried only if a suitable wire is not found.

The rules in the Connect agents knowledge base try to strike a balance between the number of particular cases they take into consideration and the likelihood of any of the particular cases leading to a perfect wire. Some examples of these special circumstances have been illustrated, but there are others. Many more can be added as the program matures.

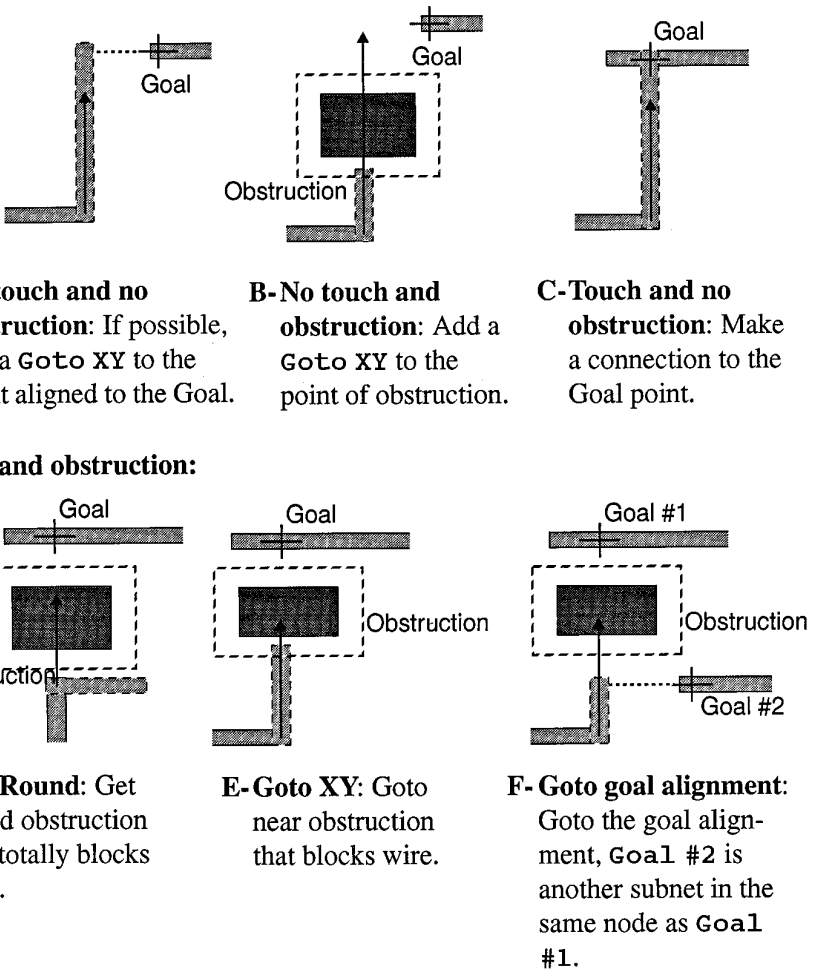


Fig. 8. Change direction operator in action.

The basic behavior of each Connect agent is to try to extend the wire it has inherited from its parent. Its job is to try all reasonable possibilities for expanding the wire. The task of the RouterExpert agent is to restrain the Connect agents in such a way that the program finds a good solution in a reasonable amount of time. The interaction of these two types of agents creates the final routing.

6. A SCALABLE SOLUTION

Figure 9 shows a set of servers working together. The Placer server refines the placement of a circuit by undertaking placement/routing cycles. In each cycle, a number of generations are run to produce a placed cell that is sent to an available Router server for routing. These cycles are repeated until a Router server produces a suitably routed design.

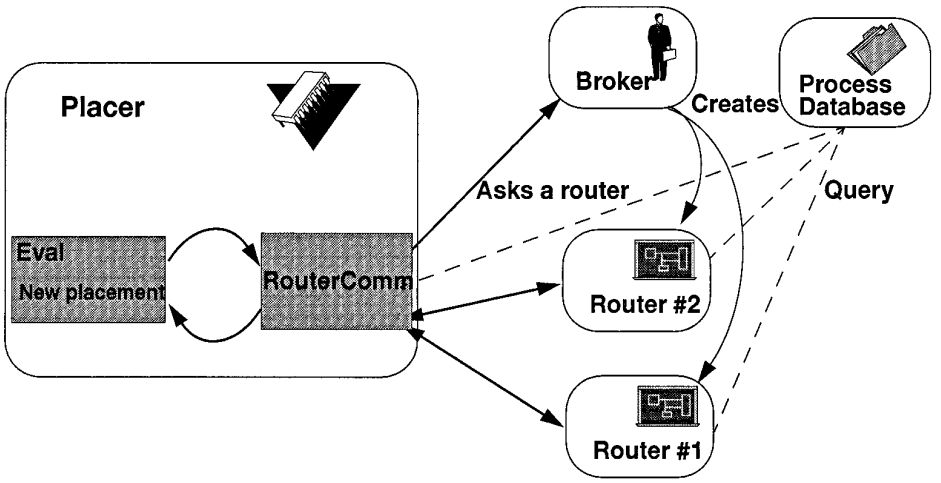


Fig. 9. The servers working together.

Many Router servers can work in parallel on different computers over a network. As a Placer server can generate placements faster than the Router servers can, a set of one Placer and many Router servers generate circuits faster than just one server doing both operations. The actual number of Router servers running at the same time is controlled by the Broker server, and will depend on the computational resources available to the program. As a scalable solution, the program can run on just one computer with just one Router server or on a network with many servers.

The Agents system was designed primarily to show the feasibility of a distributed approach to layout generation based on software agents. To test the system's ability to run in a distributed way over a network and also its scalability, the program was modified to generate ten layouts of two circuits, a BICMOS two-input nand gate, and a CMOS D latch, on each of the following computer configurations:

—*PC.* A 486 DX2 66Mhz PC with 16 megabytes of memory, running Linux;
—*One workstation.* A Sun Sparc 5 workstation with 32 megabytes of memory, running SunOs.
—*Three workstations.* Two Sun Sparc 5 and a Axil230 workstations (a Sun Sparc 10 clone), all running SunOs.
—*Eight workstations.* Two Sun Sparc 5, an Axil230, two Sun IPX, and three Sun IPC workstations, all running SunOs.

The interconnected machines used an Ethernet network with a bandwidth of 10 Mb/s. Table I shows the results of the tests, all the times are in minutes and seconds. For each circuit there are two time values for each computer configuration: *first* means the time the program took to generate the first layout (it includes setup times), and *average* means the average interval between subsequent layout generations.

Table I. Agents System Execution Times

		PC	One workstation	Three workstations	Eight workstations
BICMOS nand gate	first	4:07	2:12	1:17	0:58
	average	0:57	0:40	0:22	0:20
CMOS D latch	first	9:48	2:42	3:27	3:11
	average	5:39	4:51	1:01	1:00

The values in Table I are not precise for the performance of the program. Precise execution time is dependent on the networked machines load, at the time of the benchmark. Nevertheless, as the network was not under a large workload during the test, the results are good enough to show program scalability. The PC times mainly show that the program can run on such a configuration. Scalability really begins to show from one to three workstations, when there is a big increase in performance (with a proportional reduction in execution time) as the result of the extra power available to the program. There is a more modest increase in performance with from three to eight workstations, but then the extra workstations do not add that much power compared to the three already doing the work.

Table I shows that the Agent system is indeed able to run faster as the computing resources available to it increase. It can run using just the resources of a single PC computer, or it can run on a network of powerful workstations. More hardware buys greater velocity.

The two circuits generated in the test use two very different fabrication processes to show process independence. One is Orbit's BICMOS 2 μm double-metal double-poly process and the other is ES2's CMOS dual-metal layer 1.5 μm process. The first example circuit is the BICMOS two-input nand gate [Pucknell and Eshraghian 1994, color plate 8(a)]. The layout generated for this gate, Figure 10, is very similar to the one carried out manually [Pucknell and Eshraghian 1994, color plate 8(a)]. The generated layout is just about 7% bigger than the manual.

Figure 11 shows the layout generated for the CMOS D latch [West and Eshraghian 1993, p. 327]. The handmade symbolic layout, Fig. 11(a), was implemented in two strips of diffusion, but the generated mask layout, Fig. 11(b) had to break its Ndiff strip into three. This latch layout is small but tricky. The trick is the crossing of the clock signals, shown in the symbolic layout. The system is not currently programmed to detect this kind of transistor alignment during the column formation process. Nevertheless, the generated layout is still of a good quality. Adding rules, specifically to detect this kind of transistor alignment in the Abutted agents, could solve this problem.

7. CONCLUSION

AGENTS is a distributed system for the generation of mask-level layout of full-custom CMOS, BICMOS, and bipolar leaf cells. It is composed of four

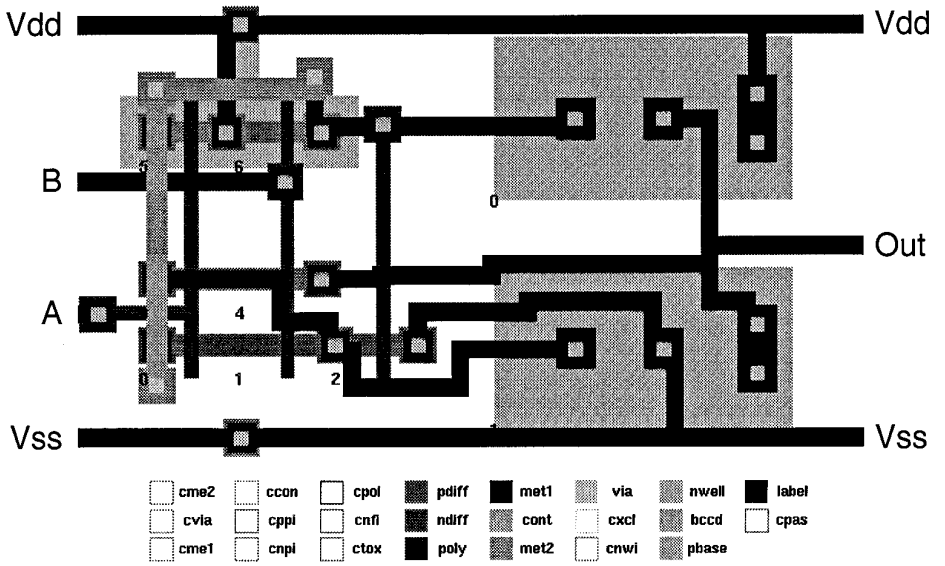


Fig. 10. BICMOS nand gate layout.

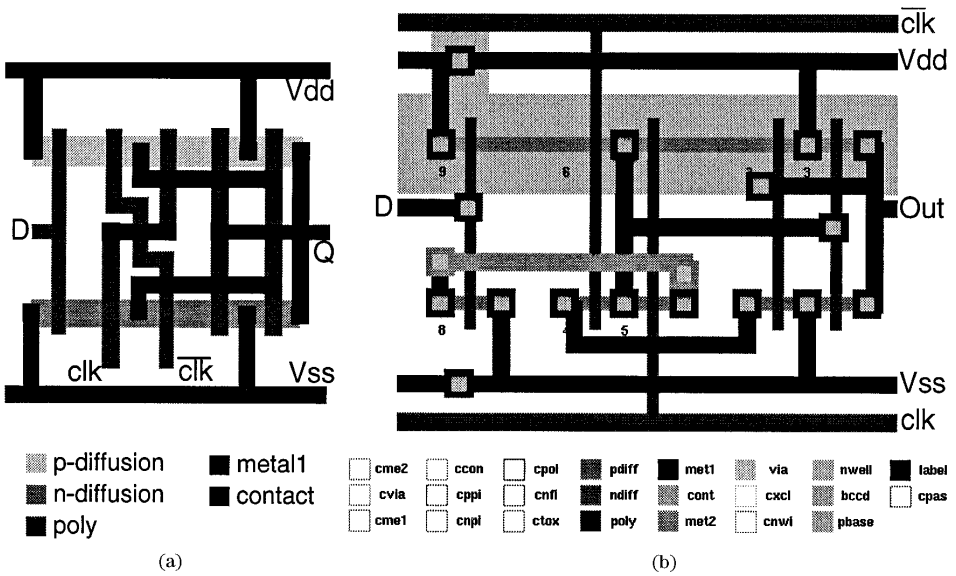


Fig. 11. CMOS latch D layout. (a) Handmade symbolic layout; (b) generated mask layout.

separate server programs: The Placer, which places components in a cell; the Router, which wires the circuits sent to it; the Database, which keeps all the information that is dependent upon the fabrication process; and the Broker, which makes the services of the other servers available. It uses a process-independent database together with a standard circuit description language (EDIF) to allow the program to be easily integrated in other design systems.

The AGENTS system was designed primarily to show the feasibility of a distributed approach to layout generation based on software agents. As a *proof of concept* prototype system it suffers from implementation problems, such as a severe memory leak, which limits the usability of the system in a production situation. Nevertheless, it does show that a distributed system approach is not only feasible but also delivers the scalability and the capacity of using the power of distributed parallel computers to increase the performance one would expect from software agents. These characteristics become ever more important as computer systems move more and more towards networks of powerful, cheap workstations.

At present, a new version of the program is being written. It will tackle the prototype's implementation problems and produce a more usable system. Among the planned new features of the system are two new agents (servers): a transistor generator and a library generator. The library generator will allow the program to generate optimized cells for use in standard cell libraries in addition to the tailor-made cells currently generated.

ACKNOWLEDGMENTS

The authors would like to acknowledge, the CNPq—National Council for Research, an agency of the Brazilian Federal Government, for financial support.

REFERENCES

ARNOLD, M. H. AND SCOTT, W. S. 1988. An interactive maze router with hints. In *Proceedings of the 25th ACM/IEEE Design Automation Conference.* ACM/IEEE, 672–676.

BROOKS, R. A. AND FLYNN, A. 1989. Fast, cheap and out of control: A robot invasion of the solar system. *J. British Interplanetary Syst. 42.*

EDIF 1984. *EDIF Specification EDIF Electronic Design Interchange Format Version 0 9 5,* EDIF Steering Committee, Nov.

FININ, T. AND WEBER, J., ET AL. 1994. *Specification of the KQML agent-communication language.* The DARPA Knowledge Sharing Initiative External Interfaces Working Group, Feb.

GENESERETH, M. R. AND KETCHPEL, S. P. 1994. Software agents. *Commun. ACM 37,* 7 (July), 48–53, 147.

HSIEH, Y-C., HWAMG, C-Y., LIN, Y-L., AND HSU, Y-C. 1991. LiB: A CMOS cell compiler. *IEEE Trans. CAD 10,* 8 (Aug.), 994–1005.

KIM, J. AND MCDERMOTT, J. 1986. Computer aids for IC design. *IEEE Software* (March), 38–47.

KOLLARITSCH, P. W. AND WESTE, N. H. E. 1985. TOPOLOGIZER: An expert system translator of transistor connectivity to symbolic cell layout. *IEEE J. Solid-State Circuits SC-20* (June), 799–804.

LIN, Y-L. S. AND GAJSKI, D. D. 1988. LES: A layout expert system. *IEEE Trans. Comput.-Aided Des. 7* (Aug.), 868–876.

MOREIRA, D. A. AND WALCZOWSKI, L. T. 1995. A leaf-cell generator for silicon compilers. *ACM OOPS Messenger 6,* 3 (July), 50–51.

MOSSA, Z., BROWN, M., AND EDWARDS, D. 1994. An application of simulated annealing to maze routing. In *Proceedings of European Design Automation Conference* (Sept.) Session D-22, SIGDA Publications on CD-ROM Compendium 1994, ACM Press, New York.

NEWELL, A. 1990. *Unified Theories of Cognition,* Harvard University Press, Cambridge, MA.

POIRIER, C. J. 1989. Excellerator: Custom CMOS leaf cell layout generator. *IEEE Trans. CAD 8,* 7 (July), 744–755.

PUCKNELL, D. A. AND ESHRAGHIAN, K. 1994. *Basic VLSI Design,* 3rd ed., Prentice Hall, London.

RIBEIRO FILHO, J. L., TRELEAVEN, P. C., AND ALIPPI, C. 1994. Genetic-algorithm programming environments. *Computer 27,* 6 (June), 28–43.

WEST, N. H. E. AND ESHRAGHIAN, K. 1993. *Principles of CMOS VLSI Design,* 2nd ed., Addison-Wesley, Reading, MA.

Received December 1995; revised July 1996; accepted October 1996