The VLSI Floorplanning Assistant

John Kingston and Robert Inder

AIAI-TR-103

March 1992

This paper was submitted to the First International Conference on the Practical Applications of Prolog, which was held in London in March 1992.

Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

Abstract

A considerable amount of effort has been put into developing knowledge-based systems to help VLSI designers. Many of the existing systems tackle problems at the design synthesis stage, which decides which components and connections are required to obtain the desired functionality for the chip. However, once the components have been selected, they have to be laid out on the chip efficiently. European Silicon Structures (ES2) found that their layout software frequently required intervention from knowledgeable humans to optimise the area taken up by the design. The Floorplanning Assistant was therefore developed to help designers to improve a layout by altering the size of the chip or the order in which components are laid out.

The Floorplanning Assistant was developed to help designers to improve a layout by altering the size of the chip or the order in which components are laid out. The Floorplanning Assistant includes an AI component, which draws on knowledge about the reasons for gaps appearing in the layout, and applies sensible resizing operations to close up these gaps. It is possible that more than one resize may be required in order to achieve an overall improvement; this is achieved using a best-first search with local optimisation.

1 Introduction

This paper describes the development of the Floorplanning Assistant, which was built by the Artificial Intelligence Applications Institute of the University of Edinburgh for European Silicon Structures (ES2). ES2's business is the design and manufacture of custom silicon chips [1], and this system aims to help their designers to revise and improve their designs.

Knowledge-based systems and VLSI Design: The task of VLSI design has prompted a considerable number of knowledge-based systems designed to tackle various stages of the problem. When a chip is designed, the required inputs and outputs are specified first, and then *synthesis* is performed. Synthesis involves specifying how the desired outputs will be achieved from the desired inputs, initially at a very abstract level, and then at more concrete levels, until eventually a pattern of the required components and their connections is arrived at. The goal of synthesis is to produce a structure which will produce a chip which will perform rapidly and accurately, and to minimise the cost of the chip. It is also beneficial if synthesis can be performed quickly and cost-effectively. To this end, several knowledge-based systems have been developed to tackle various parts of the synthesis process, either by producing an optimised version of a design, or by producing a design at a lower level of abstraction. Examples of such systems include the Logic Consultant from Trimeter Technology Corp. [4], or Design Synthesis and Design Advisor from NCR

[1] These are usually described as ASICs - Application Specific Integrated Circuits

Microelectronics [2]. Knowledge-based systems have also been applied to verifying and validating designs, allowing designs to be input at varying levels of abstraction, and storing and maintaining design information during the design process.

The Floorplanning Assistant: laying out the design: The Floorplanning Assistant, however, is designed to assist with a later stage of the design process. Once a structure for a design has been created, the components required have to be laid out on the chip. An ideal layout will be compact, since empty space increases the size, and therefore the cost of the chip. However, ES2's existing software (which is described below) does not always achieve this goal, so the designer has to step in and force some alterations the layout. This is an awkward task which requires considerable knowledge of the workings of the placement software; it is also slow, because it can only be checked by performing a full layout, which takes a couple of hours. To remedy these difficulties, the Floorplanning Assistant was developed to allow designers to make changes to the physical placement of a design and see the results displayed graphically on a screen. It also allows designers to interrogate the system to obtain information about the design.

Embedding AI technology: The Floorplanning Assistant also incorporates an AI component which it uses to suggest and implement possible improvements to the design without human intervention. It thus integrates AI into an existing software system by providing AI capabilities as one facility within a sizeable software system, which in turn performs one stage among several stages in the design process.

By supplying this AI component, the Floorplanning Assistant increases the 'power' of the current software system by a significant amount. In one of the keynote addresses at the BCS Expert Systems '88 conference, it was suggested that "if we had software that monitored itself, reported problems, suggested solutions and implemented them (once they'd been OK'd by some trusted human), we'd have a miracle on our hands" [1]. While the Floorplanning Assistant does not fulfil all these criteria, it is a step in the right direction.

2 An overview of the current system

2.1 ES2's existing software

ES2 currently use (and sell) a suite of programs for designing integrated circuits. This suite is known as the Solo 1xxx software (e.g. Solo 1200 [3]). This software works through a number of design stages, which fall into two main categories: *logical* design and *physical* design.

Logical Design: The designer works interactively with the computer to specify the circuit in terms of interconnected functional units, known as *cells*. The system provides a range of cells in a library. The designer is encouraged to structure the design, which involves identifying groups of related cells and their interconnections as single units. In defining such *design units,* it is possible to specify that they include one or more other design units, to produce a design with a tree structure. The result of this stage is a *design specification.*

Having designed the circuit, the user is able to invoke a simulation of its functioning, treating its components as ideal logic gates. This allows the user to check that the design does what is required.

Physical Design: Physical design involves two tasks: placing the design units on the chip, and, once the actual positions for the gates are available, deciding on the precise routing for the connections between them. This process produces a complete design which can be used to drive integrated circuit manufacturing equipment without further human intervention.

Once the intended physical structure of the entire circuit is known, the user can invoke a more detailed simulation of the design. This tries to take account of the physical characteristics of the devices and interconnections, and thus give a more accurate indication of the behaviour to be expected if the circuit were built.

2.2 The design problem: ordering the cells

The routing of signal paths between cells can have a tremendous influence on the performance and size (and thus cost) of the finished circuit. The process is critically dependent on the way the cells to be inter-connected are placed on the chip. Unfortunately, provably finding the optimum routing for a circuit is computationally intractable; the distance between any two points on a chip depends on what other connections are trying to fit through the same gap!

Rather than confront the enormous computational task required for a full 2-dimensional layout, the current placement system works in a simplified domain: it initially restricts itself to laying out the cells in a single unrestricted line, and attempts to optimise cell ordering at this level. This single line is then laid out sequentially on thee chip, column by column, in a zig-zag pattern. However, even within a single row of cells, the task of optimising the ordering between them is computationally intractable. Instead, the placement system makes use of the *hierarchical structure* of the design. A logical design is made up of a number of "design units" which are actually an agglomeration of smaller design units. The placement system makes use of this by ensuring that the components at each level of the hierarchy are ordered to minimise inter-connections at that level. This approach allows the placement system to produce acceptable layouts from well-structured designs constructed from logic gates which contain less than a dozen transistors.

Fixed Blocks: Advances in VLSI technology have made it imperative that the placement system is able to deal with design units containing very large numbers of cells with many and regular interconnections such as memories. The large number of interconnections means that the relative placement of these units can greatly affect the difficulty and expense of routing them. To overcome this problem, VLSI designers often hand-craft the layout of these units. The result is that these units must be treated as blocks of fixed shape and size even though they could be functionally described in terms of smaller units.

The placement system has been modified to be able to lay out these "fixed blocks", but the designs produced are often less than satisfactory. Each fixed block is treated as a single, very large cell. As such, it is fitted in to the linear array just like any other cell, and is duly placed on the chip when its turn comes. When it is being placed, each fixed block is treated as requiring one entire row which happens to be very high, occupying as much space as several 'normal' rows. If there is insufficient room in the existing column to accommodate it, the whole block is placed in the next column, which can lead to large gaps in the layout. Similarly, if the block is narrower than the column in which it is being placed, space is wasted at the sides. These two effects can lead considerable amounts of chip space being wasted.

The designer is offered some limited facilities for influencing the behaviour of the system by adding instructions to the specification of the design. These can prevent the placement system from re-ordering either particular layers or whole sections of the design, so that they will be laid out in the order defined by the designer. It is also possible to supplement these with commands to force row or column breaks. These facilities can be used to affect the positioning of fixed blocks, but the effect is only indirect, and a clear understanding of the way the placement system operates is required to use them.

The specification of the Floorplanning Assistant was that it should allow the user to specify appropriate placements of fixed blocks or other design units directly (i.e. in terms of positions of blocks), and to provide feedback on the effects of the change. It should also generate appropriate sequencing and column breaking commands automatically. A further factor was that ES2 wanted the system to be delivered as soon as was practicable; the system was seen as a stop-gap to plug a weakness in the functionality of their current software.

3 The Floorplanning Assistant

The Floorplanning Assistant is implemented in Edinburgh Prolog [2]. The 3-man team (2 developers and 1 technical manager) spent approximately 9 man months on the project. This was the first major piece of Prolog programming for both the developers.

[2]Edinburgh Prolog 1.5.04, also known as NIP. Available from the AI Applications Institute, University of Edinburgh.

3.1 Using the Floorplanning Assistant

The user of the Floorplanning Assistant first instructs the system to read in a design specification from a file. The ordering of design units, and the size of chip specified by the design specification are treated as the current *configuration* of the design. If the design passes various checks which are applied, the user is offered a number of commands which allow him to alter the current configuration. A list of commands can be seen in the menu of commands which is shown in Figure 2. Most of the commands fall into one of five categories:

- Re-ordering and positioning the design units
- Altering the hierarchy of design units
- Resizing the chip
- Obtaining information about the chip
- Storing partially-completed configurations

Examples of commands include:

- **Auto Enhance**: Invoke the AI module (described in more detail in section 5).
- **Change Column Size**: Prompts for the new width and the new height of the column, giving the old width and the old height as defaults
- **Constrain Unit**: A design unit is forced to be laid out at the top or bottom of a particular column, or immediately before or after another design unit. Various checks are made to prevent the existence of conflicting constraints.
- **Describe Column**: The chip is divided into a number of columns. This command will print out a column's size, the units that are currently in the column (noting the positions at which they start and end), and the start and end points of any gaps in the column.
- **Explode Internal Node**: An *internal node* is a design unit which is composed of other design units. This command removes the top level of the design hierarchy, so the internal node is replaced by the design units of which it is composed, in the order in which they appear in the design specification file.

These commands do not quite achieve the requirement that the Floorplanning Assistant should allow the user to specify the placements of design units directly. This is because the layout software cannot specify the physical location of design units on the chip, with the exception of the top and bottom of columns. The Floorplanning Assistant does provide the commands "Constrain Unit" and "Move Unit", which permit a designer to revise the ordering of units or to specify constraints to the top or bottom of columns. This allows designers to specify the placements of design units without too much difficulty. After each command, the Floorplanning Assistant shows the new design to the user. Because a full layout takes a long time, the Floorplanning Assistant simulates a top-level design instead.

When the user is satisfied with the current configuration, the system will write out that configuration to a *layout ordering* file, which can be passed to the placement software along with the design specification file. The system also writes out log files, and a file which describes the salient characteristics of the current configuration so that the user can restore the configuration at a later date if required.

3.2 User Interface

Whenever a command alters the current configuration, the Floorplanning Assistant 'lays out' the current configuration, and displays a diagram of the result. Two different user interfaces were developed; one is suitable for a VDU which can only display ASCII characters, and one is able to provide a graphical display on Suns.

VDU interface: The version of the system which runs on a VDU displays a short alphanumeric string for each row of each column, representing the internal name of the largest, or the most central, design unit in that row. It also displays the width of each column, the direction in which each column was laid out (up or down alternately), and any *design constraints* (see section 4.2) on units.

```
i 49   many
i 49    f 01
i 26   vf 02v
i 26   vf 02v
i 26   [f 03]
i 14   [f 03]
i 14   [f 03]
i 09   [f 03]
i 04   [f 03]

  ^      v

-40-   -65-
  1      2
```

Figure 1: An example of the VDU interface [3]

[3] The design is the same one as is shown in Figure 2

This diagram shows a chip with two columns, which are 40 and 65 gates wide, respectively. The design units are represented as **i 04**, **f 03** and so on; the **i** indicates an internal node, and the **f** indicates a fixed block. If there are several design units in a row, and no unit occupies more than 50% of the row, the word "many" is displayed. The "v"s and brackets surrounding the names of units indicate constraints on those units; the pointers at the bottom of the columns indicate whether the columns were laid out from the top or from the bottom.

Sun interface: The version which runs on a Sun console displays a Sunview [4] window. An interface with Sunview graphics was implemented especially for this project. This interface allows Prolog clauses to be used to specify that lines, text, or shaded areas should be drawn. It is trivial to write further Prolog clauses to make arbitrarily complicated combinations of these graphic primitives, such as a box which consists of four lines, or a fixed block which consists of a shaded box with a text label. The completed Floorplanning Assistant maintains a diagram showing each design unit, in its current location on the chip; empty areas of the chip are shaded.

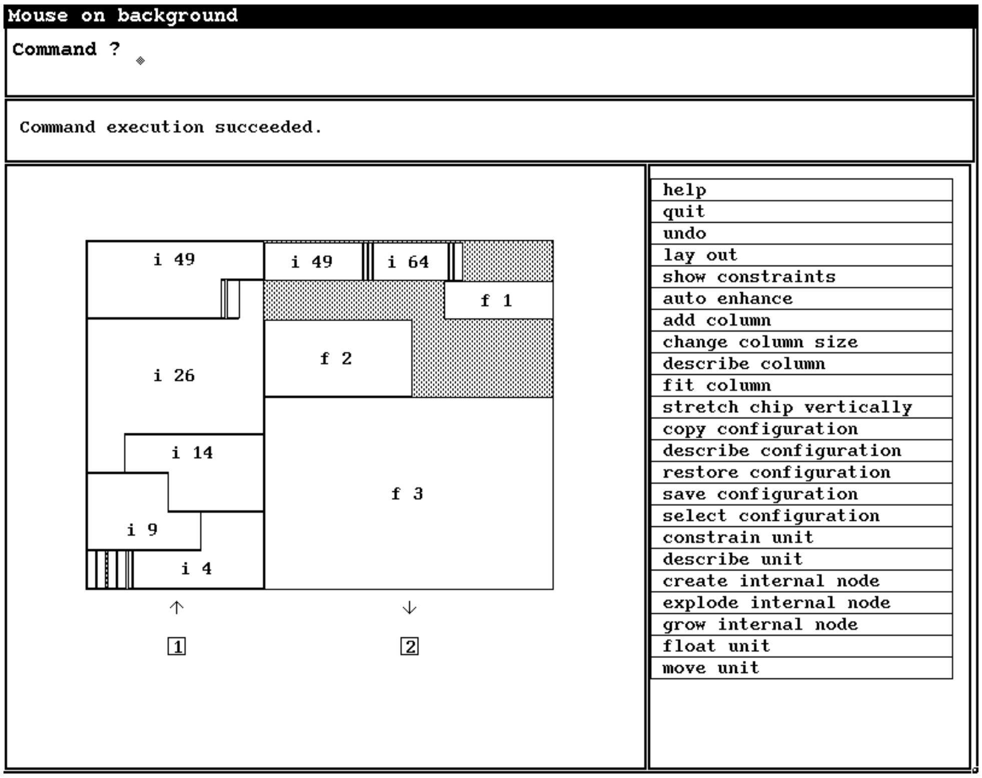

Figure 2: The Floorplanning Assistant running on a Sun console

[4]**SunView** is a trademark of Sun Microsystems Inc., Mountain View, CA

The Sun diagram is mouse-sensitive; it also offers a mouse-sensitive menu of commands (shown above, to the right of the diagram). Mouse sensitivity is achieved simply be noting the location of the mouse on the screen. A "mouse line" was implemented, which gave information about the current location of the mouse (see top left hand corner of Figure 2). This feature turned out to be surprisingly efficient; if the mouse was moved around rapidly, the mouse line was updated almost instantaneously.

4 Development

4.1 Prototype: Knowledge Craft

A prototype of the system was developed first, using Knowledge Craft [5] This was intended as a 'proof of concept' system, rather than as a fully-fledged prototype. This system made use of CRL-OPS (the OPS5-like rules in Knowledge Craft) and of Knowledge Craft's graphics component. It took about three weeks to develop; at the end of that time, the system had about twenty rules, and was able to alter the size or number of columns on the chip, revise the order in which design units were laid out, and redisplay the updated configuration.

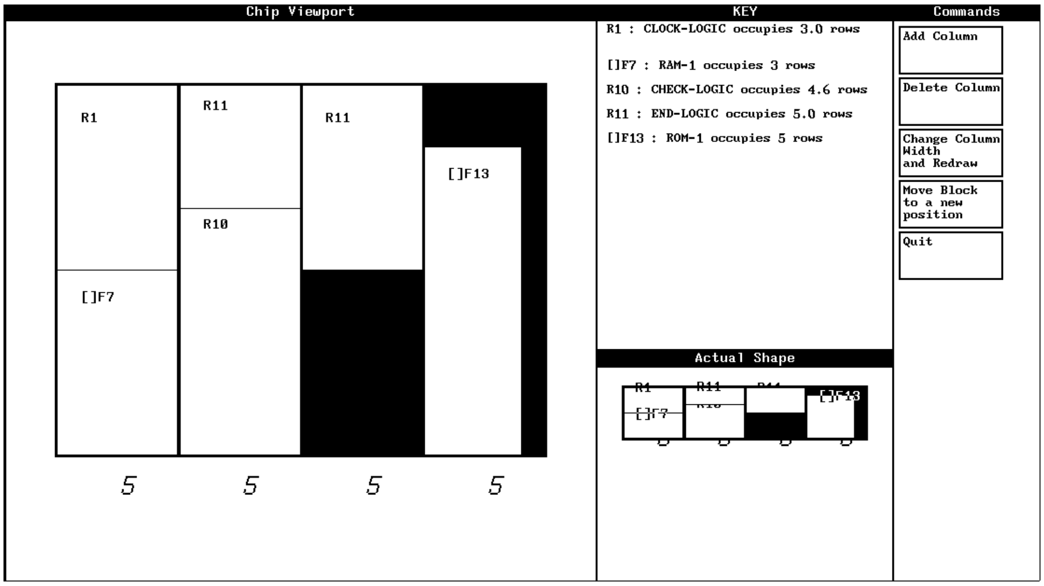

Figure 3: The interface of the Knowledge Craft prototype

[5]**Knowledge Craft** and **CRL-OPS** are trademarks of Carnegie Group Inc.

The Knowledge Craft prototype was used to discuss the functionality of the Floorplanning Assistant with ES2, and to agree on the intended functionality for the final system. It thus served as a "specification" for the delivered version of the Floorplanning Assistant.

4.2 Delivered system: Edinburgh Prolog

Attention now turned to the Prolog system. The first two tasks undertaken were to write the 'inner loop' which simulated the layout of design units on a chip, and was thus at the heart of the whole system, and to implement a simple command interpreter.

The initial idea was to lay out units sequentially, but this would have been very complicated, because the inner loop had to allow for certain design units being constrained to be laid out at the end of particular columns, and other units demanding to be laid out immediately before or after another unit. It was decided that units which were constrained to the ends of columns would be laid out first. The remaining units would be laid out sequentially; if the sequential layout should result in one unit overlapping a constrained unit which was already laid out (which was termed a *collision*), the non-constrained overlapping unit would be re-laid at the next available space. As for units being constrained to others, the inner loop treated two or more units constrained to be adjacent as if they were one design unit; if one unit was involved in a collision, the whole group would be re-laid.

The command interpreter was designed to support command completion, type checking on arguments, and input either from keyboard or mouse. Command completion in Prolog was achieved by representing commands as difference lists of difference lists of words, which allowed any possible completion to be matched. Type checking of arguments was obtained by maintaining a Prolog term for each command which noted the required type of each argument. This term was used to invoke a type-checking procedure. Input from either the keyboard or mouse was achieved by introducing a concept of *input events*. Prolog will only read input from one stream at a time, so the command interpreter was designed to scan both the keyboard input stream and the mouse input stream for a single input event. A line from the keyboard, or a click from the mouse, counted as an input event.

5 Auto Enhance: a classic AI search problem

One of the commands available to the user of the Floorplanning Assistant is the *Auto Enhance* command. This command looks at information about the location and reason for gaps on the chip (this information is recorded by the inner loop), and tries to reduce the size of these gaps. It does this by altering the size of columns.

The implementation of the Auto Enhance command turned out to be a classic task for Artificial Intelligence heuristic search techniques. There is a problem; the chip layout contains unused space. Applying operations such as widening a column will probably, but not definitely, improve the layout; hence these operations are heuristics. The number of potential heuristics is infinite (widen a column by one stage, widen it by two stages, ...). The results of applying these heuristics are difficult to predict algorithmically; however, if a layout is actually performed, the increase or decrease in the total area occupied by the layout can be used as a simple evaluation function which provides an estimate of the usefulness of each enhancement. Lastly, the inner loop provides information about the reason for each gap being present, so the system can make use of human expertise to decide on sensible ways to close up a gap

A common AI approach to tasks requiring heuristic search is to go through the necessary operations for performing the task with one of the heuristic enhancements applied. The evaluation function is then used to decide is

- The enhancement has improved affairs sufficiently
- This enhancement should be rejected and another one investigated
- Another enhancement should be applied to the revised configuration to see if the first enhancement, though apparently detrimental, has set up a situation where a second enhancement can bring major improvements. This assumes that a multiple-level search is in operation.

If possible, the system should attempt enhancements which are likely to produce the best results first.

For the Floorplanning Assistant, this approach would mean investigating each gap on the chip and deciding how the situation could be improved; making a list of commands ordered by the amount of space they were expected to save, which would alter the configuration to achieve the desired effect; and applying the first command on the list to the current configuration. The system would then perform a layout (without the graphic output), which would be examined to see if the amount of space wasted had decreased, and how much it had decreased by.

The Auto Enhance command was designed to use this technique. However, there were some further issues to be resolved. Should the search be depth-first, breadth-first, or use some other technique? How many levels of search should be used? When should the search be terminated? The story of how these issues were resolved over various versions of the system is given below.

6 Development of the Auto Enhance command

Which Commands? The first issue to be decided concerning the Auto Enhance command was exactly what it should be allowed to do to the configuration. Designers use two techniques in order to close up gaps in the layout; they either change the order in which design units should be laid out, or they change the size of columns. Should the Auto Enhance command be allowed that flexibility?

The problems associated with the routing of connections between design units have been described above. The ordering of units can have a great effect on the routing; and so, since the Floorplanning Assistant has no information about routing, it was decided that the Auto Enhance command would not be allowed to re-order design units. The system is therefore limited to improving the configuration by changing the sizes of columns. This means that the system is truly an 'assistant' to an knowledgeable designer, since the user ought to have a fair idea of the effects on routing if he chooses to revise the ordering of units.

Defining success and failure: One other issue had to be decided before any Auto Enhance command could be implemented: When should the searching stop? This involved two decisions. When should the system decide it had succeeded - i.e. what criterion was to be used to decide that an enhancement was sufficiently good to be accepted? And when should the system decide that it had failed? There is always the possibility that the last enhancement has set up a situation where a major saving could be made by performing another enhancement, just as in a game of chess, sacrificing a queen on one move may lead to a checkmate on the next move. How many successive enhancements should the system be allowed to perform on a configuration before giving up?

By empirical experimentation, it was found that the best results were obtained with two or three levels of search; that is, the system should be allowed to perform two or three successive enhancements on a configuration before giving up. In multiple-level search, the advantages of heuristic search, as opposed to exhaustive search, come to light. On an average layout, there are 3 columns, giving the system 12 possible changes that it could make (it could increase or decrease the width or height of each column). For a 2-level search, there are 144 possible changes, and 1728 possibilities for a 3-level search. Since investigating each possibility requires the whole design to be laid out, which takes about 1 second [6] , exhaustive search is not practicable for an interactive tool.

Breadth-first search: The first attempt at providing an Auto Enhance command solved the problems of deciding on success and failure by prompting the user for a desired depth of search and a success criterion. Values suggested by the documentation for the depth of search were 1, 2 or 3 levels, where a N-level search allowed the system to apply N successive enhancements to a configuration before giving up. The options for the success criterion were 'large' (a twenty per cent reduction in the space wasted by the layout), 'medium' (a ten per cent reduction in wasted space), or 'small' (any reduction in wasted space). It then attempted to

[6] The delivered system took about 1 second per layout, plus another second to redraw the diagram on Suns.

implement a breadth-first search to the number of levels specified.

The rationale for attempting a breadth-first search was that it seemed best to alter the configuration as little as possible, so that the user could identify changes that the system had made easily. However, a breadth-first search proved impracticable, because the breadth-first search algorithm would perform a layout and then gather up **all** the information about that revised configuration into a list, to be passed between Prolog goals until the time came for the system to investigate the next level of search on that configuration. This meant that the system quickly found itself maintaining several lists, each of which contained hundreds of terms.

Depth-first search: The obvious solution was to change to depth-first search. This was done, and it solved the stack overflow problems associated with breadth-first search, because the information about a revised configuration was used almost as soon as it was created, and so it did not need to be stored and carried around. However, using depth-first search made the depth of search chosen much more salient, especially as it turned out that a 3-level search on all the possible enhancements suggested by heuristic search took a prohibitively long time for a system which was supposed to be an interactive assistant. The idea of asking the user to choose a criterion for success, and a depth of search, was also appearing less and less satisfactory. Asking for a depth of search presumed that the user had some knowledge of the innards of the system, or some AI knowledge; and there seemed to be no empirical basis for stating that a "sufficiently good" improvement was one which reduced the wasted area on the chip by a certain percentage.

Best-first search: The Auto Enhance command underwent a re-structuring at this stage. Instead of looking at each type of gap, and choosing suitable widening, narrowing, heightening or lowering commands depending on the reason for the gap, an extra stage was introduced. The system now examined each gap and used its expertise to decide whether the column in which the gap occurred (or sometimes the column before it) needed to be enlarged or shrunk. These 'enlarge' and 'shrink' decisions were referred to as *goals* . The dimension to be altered (height or width) was only specified if absolutely necessary, Each goal was tagged with an estimated saving, which was approximately the area of the gap which the goal was targetted to close up. This estimate was used to sort the goals into order of expected utility, with the goals with the largest estimated savings being at the head of the list. The search thus became a "best-first" search.

Once the list of goals had been established, the system took the first of these goals, worked out which column resizing commands could fulfil the goals, and applied each command in turn. The criterion for success was changed; an enhancement was now deemed successful if it lived up to its estimated saving, since it quickly became obvious that although estimates were often optimistic, they were never unduly pessimistic. If it did not, the system carried on to the next level of investigation, or next enhancement; however, the best enhancement (or combination of enhancements) so far was stored at all times, and if this ever appeared to be better than any future enhancement (based on the estimated savings), the search was halted and the best so far was declared to be a success.

The depth of search is now encoded in a file which can be changed by the user of the system if so desired. The default depth of search for the system is 2 levels.

Multi-level depth first, pruned: The system was still very slow at performing a 3-level search, or even a 2-level search on larger designs. The number of enhancements tried therefore had to be pruned further to obtain an acceptable performance. Two pruning techniques were used, which are described below.

The first technique is to remove any enhancement whose estimated saving is too low to produce any overall improvement. If an enhancement has already been tried, and has increased the amount of wasted space (as enlarging a column might do), then there is usually little point in applying an enhancement which does not promise at least to make up for the extra wasted space, especially since the estimates are generally optimistic rather than pessimistic.

The second technique is based on the idea that, if a top-level enhancement affects a gap near the end of the layout, the rest of the chip layout is likely to be little changed, and thus there will be considerable duplication of potential enhancements when the next level of search is applied to this configuration. The system was therefore altered to maintain a list of enhancements at the previous level, and to ignore any duplicates of this list at the current level.

Using these techniques, the Floorplanning Assistant can perform a 3-level search on an average-sized design (2240 gates, 4 columns) in times ranging from 10 seconds on a design with few gaps to a minute or two for designs with many gaps. [7].

7 Evaluation against the original goals

Once ES2 had received the application and the manuals, they translated the system to run in Arity Prolog in order to incorporate the system with their PC-based Solo 1xxx software. This version used the VDU interface only. The translation simply involved the alteration of a few predicates. The system was distributed to ES2's design centres, but has not been taken further for organisational reasons.

The Floorplanning Assistant makes a significant contribution to ES2's Solo 1xxx design software. It provides a significant improvement in functionality over the previous system. It achieved and exceeded its stated objectives - the VDU interface was not part of the original specification. Although the AI component is a small part of the overall system, it plays an important part, even though the lack of routing information restricts the range of enhancements which can be performed automatically. It was implemented entirely in Edinburgh Prolog, and re-implemented in Arity Prolog, both of which proved to have sufficient functionality and sufficient speed for the task.

[7] Timings were taken on a Sun 3/260 with 16Mb of memory

8 Acknowledgements

The authors would like to thank John Dunn from ES2 for his involvement and suggestions, Ian Filby and Robert Rae of the A.I. Applications Institute for their contributions to this paper, and Leslie Kiss, formerly of the A.I. Applications Institute, for his work on developing the command interpreter.

References

[1] Partridge D. To add AI, or not to add AI? In *Proceedings of Expert Systems 88, the 8th Annual BCS SGES Technical conference*, pages 3–13, 1988.

[2] Reinkensmeyer E. Designing ASICs in a Knowledge-Based Environment. In *Design Automation Guide*, pages 14–18. NCR Microelectronics, 1989.

[3] ES2. *Solo 1200 Reference Manual.* ES2, 1988.

[4] Kim J. Artificial Intelligence helps cut ASIC design time. In *Electronic Design.* Trimeter Technologies Corp., 1987.