HAL
open science

MADMACS : a tool for the layout of regular arrays

Eric Gautrin, Laurent Perraudeau

To cite this version:

Eric Gautrin, Laurent Perraudeau. MADMACS : a tool for the layout of regular arrays. [Research Report] RR-1670, INRIA. 1992. inria-00074887

HAL Id: inria-00074887

https://inria.hal.science/inria-00074887v1

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992 25ème anniversaire

N° 1670

Programme 1
Architectures parallèles, Bases de données, Réseaux et Systèmes distribués

MADMACS : A TOOL FOR THE LAYOUT OF REGULAR ARRAYS

Eric GAUTRIN
Laurent PERRAUDEAU

Mai 1992

RR.1670

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

MADMACS: a tool for the layout of regular arrays

MADMACS : un outil pour le dessin de masques de réseaux réguliers

Eric GAUTRIN, Laurent PERRAUDEAU
IRISA, Campus de Beaulieu
35042 Rennes Cedex(France)
e-mail: gautrin@irisa.fr ou perraude@irisa.fr

Projet API, Programme 1
Publication Interne n° 641, Mars 1992, 12 pages
Soumis à : WG 10.5 IFIP Workshop on Synthesis, Generation and Portability of Library Blocks for ASIC Design

Abstract : This paper presents a tool for automatic layout of bidimensional processor arrays. The general topology of such structures consists of a processor cells array and interconnections restricted to nearest neighbors. Automatic layout of such structures can be viewed as processor cells tiling with regular routing between cells. The MADMACS design system is proposed as a tool to generate such structures. MADMACS is a complete graphics layout editor that features logical and coordinate free cursor movements. Furthermore, MADMACS provides a classical but interactive macro-command mechanism. This mechanism is particularly efficient for repetitive tasks like tiling and regular routing. Finally, MADMACS is tightly coupled to a LISP interpreter. Each MADMACS command has a functional form in the LISP language. As the interpreter evaluates an editor command function, it calls MADMACS which executes the associated command. The LISP language is also available to develop the skeleton of generators. With MADMACS coordinate free cursor movements, the designer is not concerned with the exact sizes of the manipulated objects, and so can develop re-usable code. Finally, a macro command can be saved as a new LISP function, and incorporated into a generator. The combined language and interactive approach allows one to obtain fast definitions of generators.

Résumé : Ce papier présente un outil de génération automatique de réseaux de processeurs. La topologie générale de telles structures consiste en un tableau de processeurs et des interconnexions entre voisins. Le dessin automatique des masques pour de telles structures peut être vu

comme un pavage de processeurs avec un routage régulier entre processeurs. Le système MADMACS est un outil pour la génération de réseaux. MADMACS est un éditeur graphique de dessins de masques complet qui permet des déplacements logiques du curseur et indépendants des espacements et tailles des objets. De plus, MADMACS comporte un mécanisme classique mais interactif de macro-commandes. Ce mécanisme est particulièrement efficace pour des tâches répétitives comme le pavage et le routage régulier. Finalement, MADMACS est étroitement lié à un interpréteur LISP. Chaque commande de MADMACS possède une forme fonctionnelle dans le langage LISP. Lorsque l'interpréteur évalue une telle fonction, il appelle MADMACS qui exécute la commande correspondante. Le langage LISP permet également de développer le squelette des générateurs. Avec les déplacements contextuels du curseur, le concepteur n'est plus concerné par la taille exacte des objets manipulés et peut ainsi produire du code réutilisable. Finalement, une macro-commande peut-être sauvegardée sous la forme d'une fonction LISP et incorporée dans un générateur. L'approche combinée langage et éditeur graphique permet une définition rapide de générateurs.

1 Introduction

Many signal or image processing algorithms demand regular computations which can be implemented on massively parallel architectures such as systolic arrays. With the increasing density of integrated circuits, ASIC implementation becomes an evermore attractive solution. Kung [6] shows that a regular array of processors simplifies VLSI integration: processors are identical, and interconnections are restricted to nearest neighbors. In fact, the general topology of such regular circuits can be viewed as the tiling of processor cells in an array combined with a regular routing to connect them. These interconnections can be achieved by specific routing functions (river routing, bridge routing) instead of using a general router. This regularity in tiling and routing is well suited to layout automation.

Many approaches to the layout automation have been presented in recent literature.

- In [4], an experiment with the standard cell generation of a systolic circuit is described. These tools break the topology to fit a floorplan of basic cell rows, and so loose regularity in tiling and routing. This leads to poor results in terms of area and speed.

- Datapath compilers produce very dense layouts because they make use of the regularity inherent in datapath circuits [5, 9, 14, 16]. Generally speaking, such compilers lay out the different elements of the datapath in one dimension. Thus, these tools can efficiently produce linear arrays. As regular arrays can be composed of many processors, datapath compilation results in an unbalanced block. We do not know of a compiler which is able to achieve a better ratio; this ratio improvement remains a manual task. Moreover, the linear floorplan used by the datapath compilers does not fit the bi-dimensional topology of some regular arrays.

The main problem with these tools is the fixed topology of the resulting layout. Some other approaches are more flexible.

- Many CAD systems provide a module maker to generate customized regular structures such as RAM, PLA,...These tools are mainly based on cell tiling according to a user-defined template. Routing, if necessary, must be embedded in the cells themselves or in dedicated cells. As simple routing must be generally broken into several cells to fit parametrization purpose, thus resulting in a more complicated template.

- In a procedural layout system [2, 3, 7, 8, 10, 13], the development of a generator is a programming activity. With this approach, a designer can handle any topology. However it can be very difficult to give a textual representation of some very simple graphical structure. Moreover, there is poor interaction between the text editing session and the visualization of the resulting design.

Some investigations [1, 11, 12, 15] have been made regarding the use of a procedural language interleaved with an interactive graphics editor. MADMACS is such a tool. Basically, MADMACS is a graphics layout editor that includes, in addition to standard facilities, the following powerful mechanisms:

- **Contextual cursor moving** and **graphic variables**, which allow the designer to move the cursor in a logical and coordinate free way;

- A classical but interactive **macro-command** mechanism allows the execution and memorization of commands sequences in an intelligent way. This is particularly efficient for repetitive tasks such as regular tiling and routing;

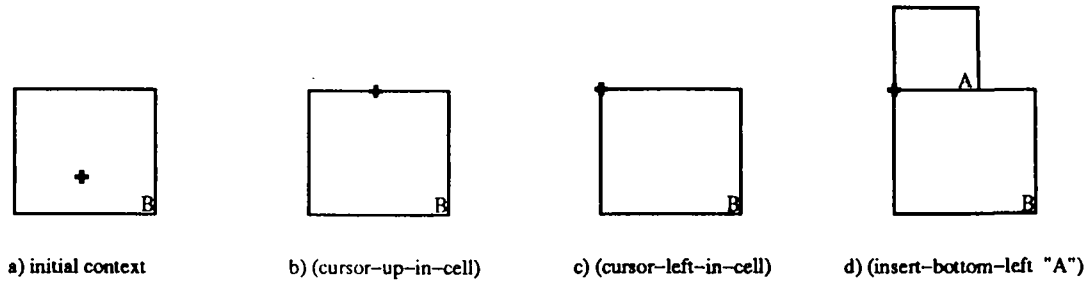

Figure 1: Example of abutment with contextual move

- MADMACS is tightly coupled to a **LISP interpreter**. Each MADMACS command has a functional form in the LISP language. As the interpreter evaluates an editor command function, it calls MADMACS which executes the associated command and returns an execution status.

This paper gives an overview of the MADMACS system. The basic concepts are presented in section 2. An example of linear systolic array is detailed in section 3. Finally, the implementation of MADMACS is presented in the last section.

2 MADMACS

2.1 Manipulated objects

Three types of objects are used in the MADMACS system.

- A **rectangle** is a piece of a layer and can be created, selected, moved, stretched, colored, deleted and named;
- A **figure** is a collection of rectangles and/or instances of figures. This type of object is used to manage the hierarchy. A figure can be created, selected, moved, rotated, mirrored, deleted, named. Moreover, the designer can move within the hierarchy;
- A **connector** can be associated with a figure. It is a special named rectangle that represents input/outputs.

2.2 Contextual cursor movement

Like many graphics editors, MADMACS offers facilities for graphic cursor motion: displacement of N λ in any direction; snap to absolute coordinates. In addition, the system provides logical and coordinate free cursor movements commands.

- move to a edge of the current cell[1], such as (cursor-left-in-cell),
- move to a neighboring cell, such as (cursor-to-upper-cell),
- align with a box edge, such as (align-with-the-right-cell).

These commands are size-independent and rely on the local context of the layout around the cursor location. Therefore, they allow the designer to ignore the absolute coordinates of layout pieces and to define size-free command sequences. For example,

[1]The current cell is defined by the cursor location.

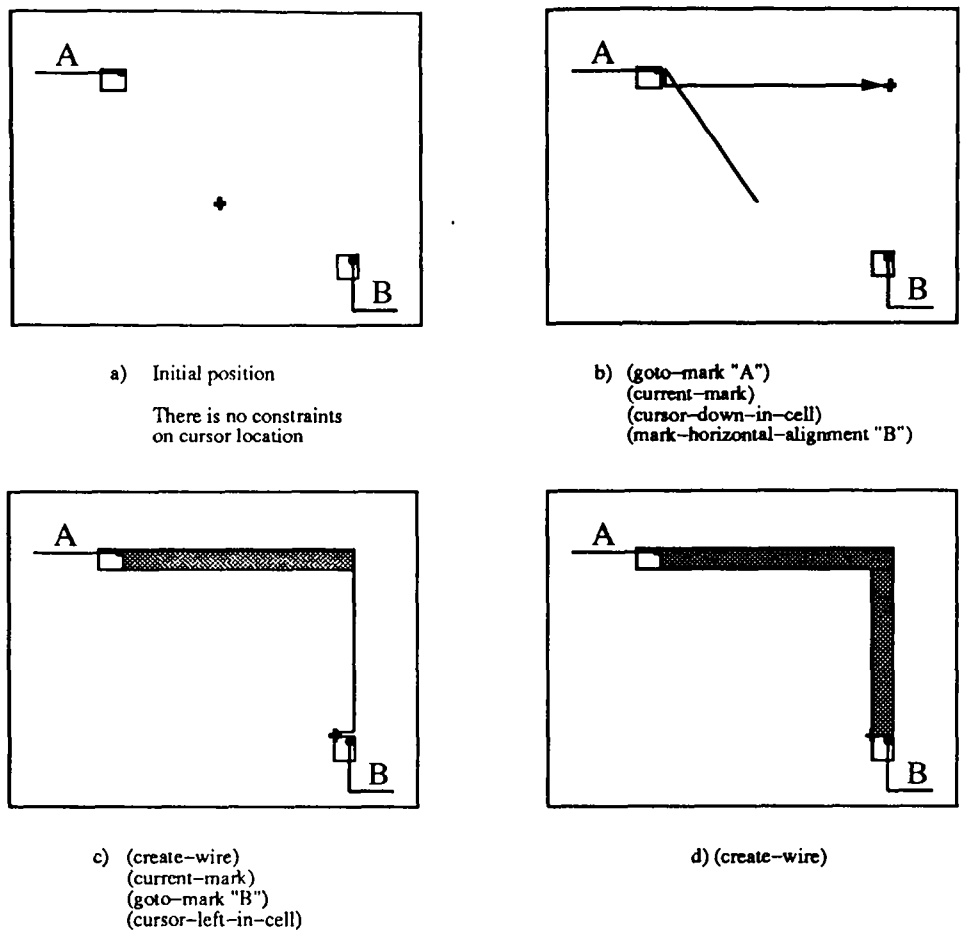

Figure 2: Creation of a right angle connection

to abut a cell A on the top of a cell B with left edges alignment, it requires only three commands. This command sequence is illustrated in figure 1. In fact, this sequence performs the abutment of cell "A" with any cell. If the cell name is a parameter, the sequence becomes even more general. Generalization is achieved with a LISP function and will be explained later.

2.3 Graphic variable

A designer will frequently return the cursor to the same location. The coordinates of this position can be identified by a named graphic variable. MADMACS provides commands for direct cursor movement such as (goto-mark "BEGIN"), or cursor alignment commands to a graphic variable such as (mark-horizontal-alignment "BEGIN"). In particular, these last commands are very useful for laying out a manhattan wire between two rectangles. The rectangles are identified by graphic variables A et B. One right angle connection is achieved by the command sequence illustrated in figure 2. Note that a wire creation needs two reference points: the current mark and the actual cursor location.

Since graphic variables value can be modified, the previous sequence can be used to connect two N bit busses A and B in the same routing style:

1. Identification of the first bit line of each bus by graphic variables A and B;

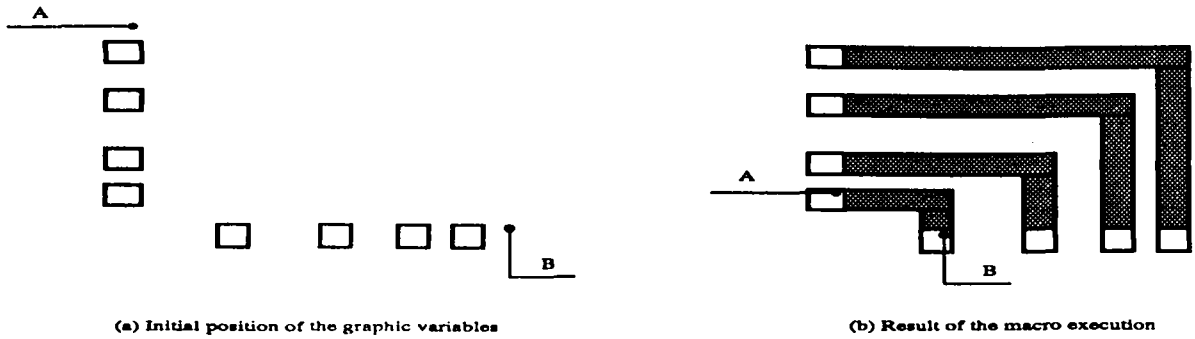

Figure 3: Multiple executions of *(right-angle-connection)* macro

2. Iteration on each bit:

   Execution of the sequence;

   Transfer of graphic variables A and B to the next bus line.

This example of bus connection leads naturally to the notion of macro-command.

2.4 Macro-command

Like many systems, MADMACS can memorize a sequence of commands called a macro-command (macro for short). To build a macro, the user enters the Macro Learning mode. The system executes and stores each command issued. The macro construction is finished when the user leaves the Macro Learning mode. The macro is then available as a single new command. Using contextual cursor movement and graphic variables, it can be coordinate free, and so can be executed in different locations as long as the execution context remains similar to the definition one.

Macros are mainly used for repetitive tasks where the same sequence must be executed several times. For example, the connection between two sets of four connectors, as shown in figure 3, is performed by four successive executions of the following macro:

```
(goto-mark "A")
(cursor-to-bottom-cell)
(put-mark "A")
(goto-mark "B")
(cursor-to-left-cell)
(put-mark "B")
(make-a-right-angle-connection)
```

The *(make-a-right-angle-connection)* command performs a right angle connection as illustrated in figure 2.

The macro mechanism is powerful, but does not provide execution control. For example, if the macro is executed five times to connect two four bit busses, the layout result will be incorrect. To introduce control, the macro is saved in its equivalent LISP form and control statements are added to this form, as illustrated in next section.

2.5 LISP interpreter

MADMACS provides the designer with an interface through a high level interpreted language (LISP). Each MADMACS command has a functional form in the LISP language. As

the interpreter evaluates an editor command function, it calls MADMACS which executes the associated command and returns an execution status. For instance, (cursor-to-upper-cell) returns 0 if there is a cell above, 1 otherwise.

With LISP, a designer can define very powerful functions with parameters and execution control. For example, the following function is a generalization of the previous macro. The *nbconnector* parameter represents the width of the bus. The previous macro uses two marks. In this function, one verifies whether these marks exist, and whether the next connector exists before moving a mark.

```
(defun connection (nbconnector)
   (COND ( (= nbconnector 0) t)
         ( ( AND (= (goto-mark "A") 0) (= (cursor-to-bottom-cell) 0))
             (put-mark "A")
             (COND ( (AND (= (goto-mark "B") 0) (= (cursor-to-left-cell) 0))
                     (put-mark "B")
                     (make-a-right-angle-connection)
                     (connection (- nbconnect 1))
                   )
                   ( t NIL)
             )
         )
         ( t NIL)
   )
)
```

This example illustrates the generalization of a macro in a LISP function by addition of parameters and control. The same methodology is used to define generators, one example of which is presented in the next section.

3 A linear array generator

To achieve a better ratio for the layout, the array is broken into several columns of processors. The general floorplan is presented in figure 4: processor cells are abutted in columns, data signals are routed vertically from column to column in metal 2 layer, and control signals are distributed horizontally through the columns in metal 1 layer. Note that the last column can have fewer cells than the others.

The linear array generator uses several parameters: **nbcell** is the number of processor cells, **nbcolumns** and nbrows are respectively the number of columns and rows of the resulting layout. A processor cell is considered as a figure with connectors: **nbdata** is the number of data signal connectors, and **nbcontrol** the number of control signal connectors. Finally, the generator uses four parameters: width and spacing parameters for both metal 1 and 2 layers.

3.1 Processor cell constraints.

Data signals run across the cells for vertical abutment without routing. Control signals run horizontally through cells for distribution to a next column. The generator considers a processor cell as a figure with connectors on each side: vertical connectors for control and horizontal connectors for data.

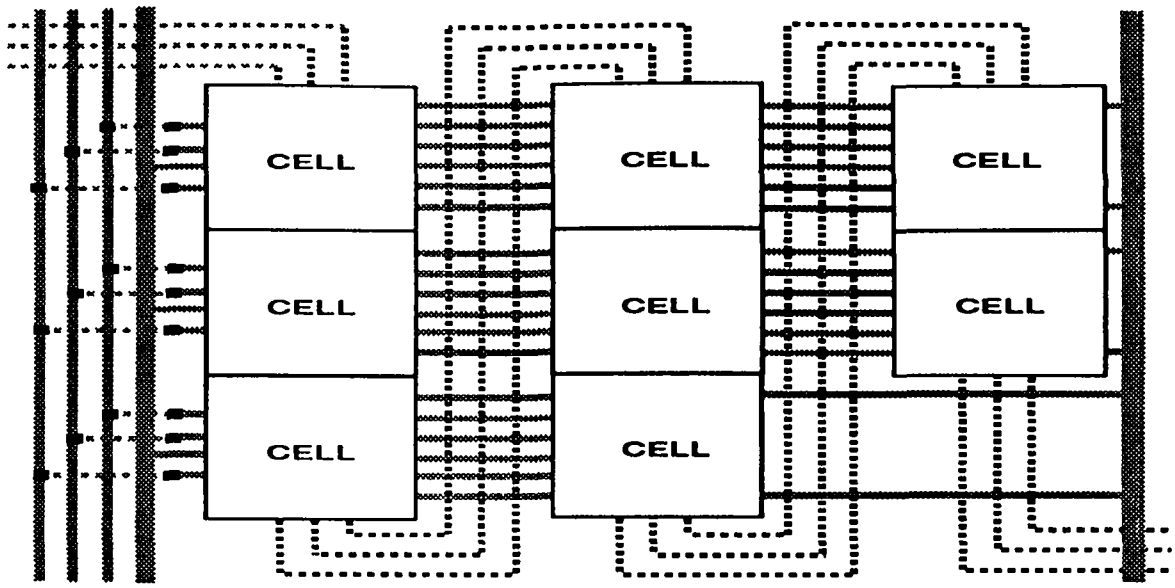

Figure 4: Linear array floorplan

3.2 Processor cell tiling.

A LISP function is defined to abut automatically a cell on top of one another. It is equivalent to the following command sequence.

```
(defun top-left-abut (cell-name)
    (cursor-up-in-cell)
    (cursor-left-in-cell)
    (insert-bottom-left cell-name)
)
```

This simple LISP function can be generalized for the creation of a column:

```
(defun create-column (nbcell cell-name)
    (COND ( (≠ nbcell 0)
            (top-left-abut cell-name)
            (create-column (- nbcell 1) cell-name)
          )
          (t NIL)
    )
)
```

There are different ways to deal with the processor column spacing. In this example, it is computed from **nbdata** parameter and width-spacing parameters.

3.3 Control signal routing.

There are two different routing strategies for the control signals: distribution between the columns and connection to the global control lines.

Distribution: Processor cell constraints guarantee that a control connector and its opposite on next column are on the same line. Distribution is accomplished by an horizontal wire from a connector to the next column. The following function creates an horizontal wire for distribution:

```
(defun horizontal-wire
    (connector-down)
    (current-mark)
    (cursor-to-right-cell)
    (cursor-down-in-cell)
    (create-wire)
    (cursor-to-left-cell)
)
```

As shown previously, this simple function can be generalized to perform multiple horizontal connections.

Global connection: The first step for these connections is to create vertical global lines on the left of the array (except for **vdd**). A graphic variable is associated with each line. Indeed, a correspondence between connectors and graphic variable names is required. The cursor being located on the connector, the wire is laid out through an horizontal alignment to the global line graphic variable.

```
(defun global-connection
    (connector-down)
    (COND ( (≠ (name-connector) "vdd")
            (current-mark)
            (cursor-down-in-cell)
            (mark-horizontal-alignment (name-connector))
            (create-wire)
            (goto-current-mark)
          )
          (t nil)
    )
)
```

If one adds more control, this function can perform two different types of wire depending on the connector name. **gnd** wires are routed in metal 1 layer, while others must use metal 2 layer. A similar function is used for **vdd** connections.

3.4 Data signal routing.

Data signal routing is achieved using three routing functions: input, inter-column and output. Input and output routing functions are similar to the routing example presented in section 2. The inter-column routing function is based on the same methodology, but requires four graphic variables (one for each turn of a connection). This routing is done in metal 2 layer, respecting minimal width and spacing constraints.

All these functions and the previous ones have been built using the same methodology. First the designer creates a macro (using contextual cursor movements and graphic variables). When the macro is validated, he saves it in a LISP form, and adds control and parameters to generalize it. This approach leads to the fast definition of generators. In a few hours, one can create an initial version of a linear array generator.

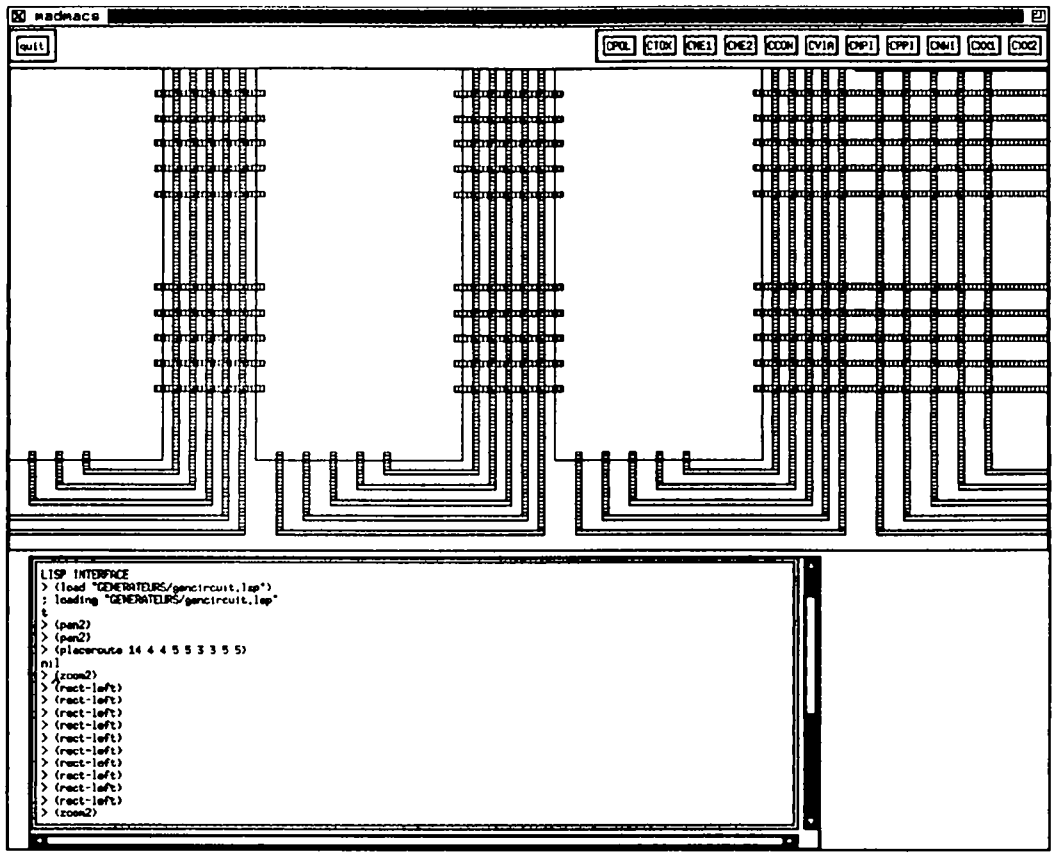

Figure 5: MADMACS interface

4 Implementation

MADMACS is written in the object-oriented language C++. Object oriented languages [17] are well suited for object manipulation such as MADMACS operations. C++ allows one to easily develop new commands and to reuse code. The MADMACS interface is constructed on top of the X Window system, and illustrated in figure 5:

- At the top, there are a set of buttons to select a layer, or to quit the application.
- The graphics window displays the layout and the effects of MADMACS commands.
- Through the textual window the designer can interact with the LISP interpreter.

Commands can be issued in two ways: in their LISP form through the textual window, or directly in the graphics window with a keystroke. This technique, used in text editors like Emacs, provides a very high interactivity. The main problem is that the user has to memorize many single key commands. However in most cases, the key associated with a command is related to its name and therefore easy to remember. Our experience has proven that such an interface is not a drawback. A designer does not use the MADMACS system to design cells as with a classical graphics editor. MADMACS is dedicated to the development of generators. A designer works with the graphic interface to interactively capture his methodology for structure construction. Then, most of the time is spent with a text editor, writing a LISP generator.

5 Conclusion

In this paper, we have described a highly interactive circuit layout system. The system combines in an intelligent way, graphics and programming facilities. An important feature is the possibility of defining size-independent command sequences. With the interactive macro mechanism, the designer is able to memorize such a sequence as a new command. Then, when validated, the sequence is saved in a LISP form and can be generalized by addition of control and parameters.

This approach has several advantages. The ability to define size-independent functions is particularly important. The designer can define libraries of functions, and use them to define new generators with less effort. Furthermore, if it is difficult to give a textual version of graphical structure or construction, the designer can handle it easily with the interactive macro mechanism. Indeed, it is a good method for capturing the designer's methodology.

References

[1] J. Batali, N. Mayle, H. Shrobe, G. Sussman, D. Weisse. The DPL/Daedalus Design Environment. In *VLSI'81*, John P. Gray ed., Academic Press, pages 182–193, 1981.

[2] JM. Berge, L. O. Donzelle, J. Rouillard, D. Rouquier. LOF. Technical note, CNET-FRANCE, 1985.

[3] M. R. Burich. Programming language makes silicon compilation a tailored affair. Electronic Design, December 1985.

[4] C. Dezan, E. Gautrin, H. Le Verge, and P. Quinton. Synthesis of systolic arrays by equation transformations. In *ASAP 91*, Barcelone, Spain, September 1991.

[5] P. Drenth and C. Strolenberg. Datapath layout generation with in-the-cell routing and optimal column resequencing. In *Euro ASIC' 91*, pages 373–376, IEEE, May 1991.

[6] H.T. Kung. Why systolic architectures? *IEEE Computer*, Vol 15(n° 1):pages 37, 1982.

[7] J. A. Lewis, A. A. Berlin, A. J. Kuchinsky, P. K. Yip. Integrated Circuit Procedural Language. Hewlett-Packard Journal, pages 4–10, June 1986.

[8] R.J. Lipton, S.C. North, R. Sedgewick, J. Valdes, G. Vijayan. ALI: a Procedural Language to Describe VLSI Layouts. In *ACM IEEE 19th Design Automation Conference Proceedings*, Las Vegas, Nevada, pages 467–474, June 1982.

[9] T. Mashburn, I. Lui, R. Brown, D. Cheung, G. Lum, and P. Cheng. Datapath: a CMOS data path silicon assembler. In *23rd Design Automation Conference*, pages 722–729, 1986.

[10] R. Mathews, J. Newkirk, P. Eichenberger. A Target Language for Silicon Compilers. In *Digest of Papers COMPCON82*, San Francisco, California, pages 349–353, February 1982.

[11] R.N. Mayo, J.K. Ousterhout. Pictures with Parentheses: Combining Graphics and Procedures in a VLSI Layout Tool. In *ACM IEEE 20th Design Automation Conference Proceedings*, Miami Beach, pages 270–276, June 1983.

[12] P. Petit. Chipmonk: An Interactive VLSI Layout Tool. In *Digest of papers COMPCON82: High Technology in the Information Industry*, pages 302–304, February 1982.

[13] S. Sastry, S. Klein. Plates: a Metric-Free VLSI Layout Language. In *Proceeding Conference on Advanced Research in VLSI*, Cambridge, Massachusetts, pages 165–174, January 1982.

[14] H.E. Shrobe. The datapath generator. In *CompCon82 High Technology in the Information Industry*, pages 340–344, IEEE Computer Society, 1982.

[15] S. Trimberger. Combining Graphics and a Layout Language in a Simple Interactive System. In *ACM IEEE 18th Design Automation Conference*, Nashville, pages 234–239, June 1981.

[16] VLSI Technology Inc. *Datapath Compiler.* Technical Report, VLSI Technology Inc., San Jose, CA, USA, June 1989.

[17] W. Wolf. Object-Oriented Programming for CAD. In *IEEE Design and Test of Computers*, pages 35-42, March 1991.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 632 L-STABLE PARALLEL ONE-BLOCK METHODS FOR ORDINARY DIFFERENTIAL EQUATIONS
Philippe CHARTIER, Bernard PHILIPPE
Janvier 1992, 28 pages.

PI 633 ON EFFICIENT CHARACTERIZING SOLUTIONS OF LINEAR DIOPHANTINE EQUATIONS AND ITS APPLICATION TO DATA DEPENDENCE ANALYSIS
Christine EISENBEIS, Olivier TEMAM, Harry WIJSHOFF
Janvier 1992, 22 pages.

PI 634 UN NOYAU DE SYSTEME REPARTI POUR LES APPLICATIONS GEREES PAR UN TEMPS VIRTUEL
Philippe INGELS, Carlos MAZIERO, Michel RAYNAL
Janvier 1992, 20 pages.

PI 635 A NOTE ON CHERNIKOVA'S ALGORITHM
Hervé LE VERGE
Février 1992, 28 pages.

PI 636 ENSEIGNER LA TYPOGRAPHIE NUMERIQUE
Jacques ANDRE, Roger D. HERSCH
Février 1992, 26 pages.

PI 637 TRADE-OFFS BETWEEN SHARED VIRTUAL MEMORY AND MESSAGE PASSING ON AN IPSC/2 HYPERCUBE
Thierry PRIOL, Zakaria LAHJOMRI
Février 1992, 26 pages.

PI 638 RUPTURES ET CONTINUITES DANS UN CHANGEMENT DE SYSTEME TECHNIQUE
Alan MARSHALL
Mars 1992, 510 pages.

PI 639 EFFICIENT LINEAR SYSTOLIC ARRAY FOR THE KNAPSACK PROBLEM
Rumen ANDONOV, Patrice QUINTON
Mars 1992, 20 pages.

PI 640 TOWARDS THE RECONSTRUCTION OF POSET
Dieter KRATSCH, Jean-Xavier RAMPON
Mars 1992, 22 pages.

PI 641 MADMACS : A TOOL FOR THE LAYOUT OF REGULAR ARRAYS
Eric GAUTRIN, Laurent PERRAUDEAU
Mars 1992, 12 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399