MADMACS : un outil de placement et routage pour le dessin de masques de réseaux réguliers

Eric Gautrin, Laurent Perraudeau, Oumarou Sié

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° 1671

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

MADMACS : UN OUTIL DE PLACEMENT ET ROUTAGE POUR LE DESSIN DE MASQUES DE RÉSEAUX RÉGULIERS

Eric GAUTRIN
Laurent PERRAUDEAU
Oumarou SIE

Mai 1992

RR_1671

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: un outil de placement et routage pour le dessin de masques de réseaux réguliers

MADMACS : a placement and routing tool for the layout of regular arrays

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

Projet API, Programme 1

Soumis au : 1er Colloque Africain
sur la Recherche en Informatique

Publication Interne n° 653 - Avril 1992 - 16 pages

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 requires two steps. First the basic elements of a processor are assembled and interconnected to obtain a cell. Then, the second step consists of processor cells tiling with regular routing between them. The MADMACS design system is proposed as a tool to generate such structures according to these two steps. To start with, a placement tool (MadPlace) produces a LISP function for assembling and routing the basic elements of a processor cell from a transfert register level description. After that, the MADMACS system is used to develop generators in order to assemble and route the processors into the array. 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 associates 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 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 automatique pour le dessin de masques de réseaux de processeurs. La topologie générale de ces structures consiste en un ensemble de processeurs interconnectés localement. Une automatisation du dessin des masques nécessite deux étapes: un assemblage d'opérateurs de base interconnectés pour générer un processeur, puis un pavage et routage de ces processeurs pour construire le réseau. Le système de conception MADMACS se propose de générer automatiquement de telles structures. Dans un premier temps, un outil de placement (MadPlace) produit une fonction LISP d'assemblage et d'interconnexions des opérateurs de base d'un processeur à partir d'une description de niveau portes logiques. Dans un second temps, le système MADMACS permet de développer des générateurs pour assembler et router les processeurs en réseau. MADMACS est un éditeur complet de dessins de masques qui permet des déplacements logiques indépendants des coordonnées géométriques. De plus, il offre un mécanisme interactif de macro commandes, particulièrement efficace pour les tâches répétitives comme le pavage ou le routage réguliers. Enfin, MADMACS est couplé à un interpréteur LISP. Chaque commande de MADMACS a une forme fonctionnelle dans ce langage. Lorsque l'interpréteur évalue une fonction de l'éditeur, il appelle MADMACS qui exécute la commande associée. Le langage LISP est utilisé pour développer le squelette des générateurs. Grâce aux déplacements logiques du curseur, le concepteur ne se préoccupe pas des tailles exactes des objets manipulés et développe ainsi un code réutilisable. Enfin, une macro commande peut être sauvegardée comme une nouvelle fonction LISP et incorporée dans un générateur. Cette approche combinée, langage et éditeur graphique, permet le développement rapide de générateurs. 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

De nombreux algorithmes de traitement du signal ou de l'image nécessitent des calculs réguliers qui peuvent être implantés sur des architectures massivement parallèles telles que les réseaux systoliques. Avec l'accroissement de la densité d'intégration, la réalisation d'un circuit dédié à une application spécifique (ASIC) est une solution de plus en plus attractive. Kung [8] a montré qu'un réseau de processeurs simplifie l'intégration VLSI : les processeurs sont identiques et les interconnexions sont locales. La génération automatique de telles structures nécessite deux étapes :

- Un processeur est un assemblage linéaire ou bidimensionnel d'opérateurs de base avec routage des signaux de données et de contrôle. Ce routage est fortement contraint car il doit conserver la régularité du niveau supérieur ;
- Le réseau est un pavage de processeurs avec un routage régulier pour les interconnecter. Ces interconnexions sont effectuées par des fonctions de routage (routage rivière, routage pont, ...) plutôt que par un routeur général. Cette régularité du pavage et du routage est une propriété très intéressante pour l'automatisation des dessins de masques.

Plusieurs approches pour l'automatisation des dessins de masques ont fait l'objet de présentations récentes.

- Dans [4], une approche pour la génération automatique de circuits systoliques à base de cellules précaractérisées est présentée. Les outils utilisés mettent à plat la hiérarchie et suivent un plan de masse constitué de lignes de cellules de base. De ce fait, la régularité du pavage et du routage est perdue. Les résultats ainsi obtenus sont décevants en termes de surface occupée et de vitesse de fonctionnement.
- Les compilateurs de chemins de données produisent des dessins de masques très compacts car ils tirent partie de l'organisation des opérateurs en tranches de bit [5, 14, 22, 24]. Typiquement, ces outils placent les opérateurs sur une seule dimension. Cette approche pourrait convenir à des réseaux linéaires, mais ces derniers étant souvent composés de beaucoup de processeurs, il en résulte un bloc beaucoup plus long que haut. De plus, ces compilateurs sont peu appropriés à des topologies bidimensionnelles.

Ces outils imposent un plan de masse peu approprié aux topologies en réseaux. D'autres approches fournissent plus de souplesse dans le choix du plan de masse.

- Certains systèmes de CAO offrent des outils de génération de structures régulières comme les RAMs, les PLAs, ... Ces outils se contentent d'assembler sans routage des cellules de base suivant un patron défini par le concepteur. Cette approche permet une grande souplesse dans le choix du plan de masse, mais aucune aide n'est fournie à l'utilisateur pour la génération d'un processeur qui demeure un assemblage manuel.
- Pour développer des générateurs de structures régulières, certains systèmes proposent uniquement un langage de programmation [2, 3, 11, 12, 15, 21]. D'autres associent au langage un éditeur graphique interactif [1, 6, 16, 19, 23]. Avec ces approches, le concepteur peut adopter n'importe quel plan de masse, mais comme pour les outils d'assemblage, aucune aide n'est apportée pour la génération d'un processeur.

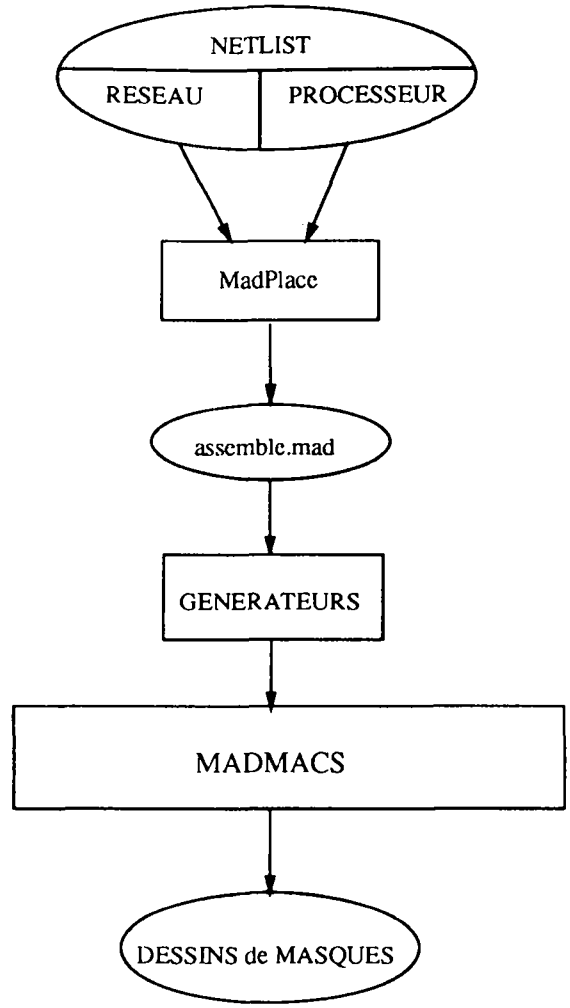


Figure 1: Vue globale de MADMACS.

Le système MADMACS est développé pour la génération de réseaux réguliers. Pour le niveau processeur, une approche compilation de chemins de données est adoptée. Cependant, un regard particulier est porté sur le routage interne pour conserver la régularité du niveau supérieur. L'assemblage et le routage du réseau sont réalisés par l'éditeur graphique MADMACS couplé avec un interpréteur LISP. Cet outil offre une grande souplesse pour la définition du plan de masse et permet le développement rapide de générateurs.

La première section de ce papier donne un aperçu général du système MADMACS. Les concepts de base de l'éditeur sont présentés dans la section 3. La section 4 traite de la génération d'un processeur. Un exemple de générateur pour réseau linéaire est détaillé dans la section 5. La dernière section traite de l'implantation de MADMACS.

2 Le système MADMACS

Notre système se place en aval du système Alpha du Centaur [20]. A partir d'un système d'équations récurrentes uniformes, Alpha donne en résultat une description de niveau portes logiques de l'architecture régulière. En fait, cette description contient deux types d'informations: les unes relatives au processeur lui-même et les autres au réseau. La

génération d'un réseau de processeurs s'effectue en deux étapes comme l'illustre la figure 1.

La première étape génère un processeur à partir de sa description niveau portes. A chaque composant de cette description correspond une cellule de base en librairie. L'outil de placement MadPlace cherche un ordonnancement linéaire des composants optimisant la coupe maximale du graphe ou hypergraphe modélisant le processeur. Dans sa version actuelle, cet outil ne produit pas de placement bidimensionnel. MadPlace fournit en résultat une fonction LISP d'assemblage et de routage des différents composants. Cette fonction est ensuite évaluée par l'interpréteur LISP ; la construction des dessins de masques est visualisée par l'éditeur graphique de MADMACS.

La seconde étape est l'assemblage et le routage des processeurs en réseau. Deux approches sont possibles pour le concepteur : utiliser un générateur prédéfini, ou développer son propre outil. Comme pour une topologie donnée de réseaux plusieurs plans de masse sont possibles, il est nécessaire d'offrir à un concepteur un environnement pour le développement de générateurs. Ceux-ci sont des fonctions LISP pour l'assemblage et le routage. Leur évaluation fournit en résultat un dessin de masques de réseau.

3 L'éditeur MADMACS

3.1 Objets manipulés

Trois types d'objets sont utilisés dans le système MADMACS.

- Un **rectangle** représente un morceau de couche technologique ;
- Une **figure** est un ensemble de rectangles et/ou d'instances de figures. Ce type d'objet permet de gérer la hiérarchie ;
- Un **connecteur** peut être associé à une figure. C'est un rectangle spécial possédant un nom et représentant les entrées/sorties d'une figure.

3.2 Déplacement contextuel du curseur

Comme la plupart des éditeurs, MADMACS offre des facilités de déplacements du curseur : déplacement de N λ dans n'importe quelle direction, positionnement à des coordonnées absolues. De plus, le système offre des commandes de déplacements logiques :

- Déplacement sur le bord de l'objet courant[1], comme (cursor-left-in-cell),
- Déplacement à un objet voisin, comme (cursor-to-upper-cell),
- Alignement avec un objet, comme (align-with-right-cell).

Ces commandes sont indépendantes des tailles et espacements entre objets. Par exemple, pour abouter une cellule A au dessus d'une cellule B en alignant leurs côtés gauches, seulement trois commandes sont nécessaires comme l'illustre la figure 2. En fait, cette séquence réalise l'aboutement d'une cellule A avec n'importe quelle cellule. De plus, en remplaçant le nom de la cellule A par un paramètre, la même séquence se généralise en une fonction LISP comme il est expliqué par la suite.

[1] L'objet courant est défini par la position du curseur.

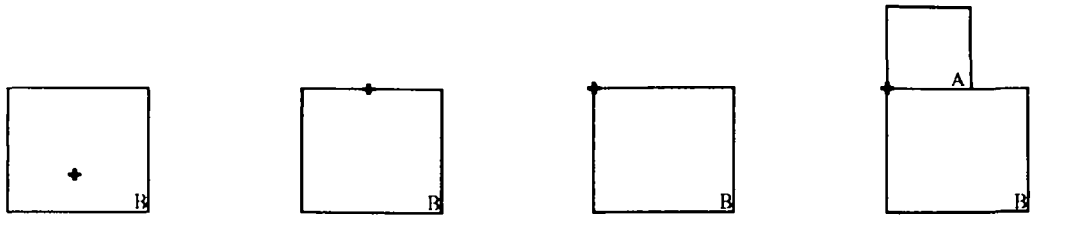


Figure 2: Exemple d'aboutement de cellules avec déplacements contextuels.

3.3 Variable graphique

Un concepteur doit fréquemment déplacer le curseur au même point. Les coordonnées de ce point peuvent être mémorisées par une variable graphique nommée. MADMACS offre ainsi au concepteur des commandes pour le déplacement direct du curseur à ce point : (goto-mark "BEGIN"), ou des commandes pour aligner le curseur par rapport à ce point : par exemple, (mark-horizontal-alignment "BEGIN"). Ces dernières commandes sont particulièrement utiles pour dessiner une connexion avec angle droit entre deux connecteurs identifiés par les variables graphiques A et B. La figure 3 illustre la séquence de commandes qui réalise cette connexion[2].

En déplaçant successivement les variables graphiques, cette séquence de commandes réalise un bus avec angle droit entre des connecteurs disposés sur des axes perpendiculaires :

1. Marquage des premiers connecteurs par les variables graphiques A et B;
2. Pour chaque connecteur faire :

   Exécution de la séquence ;

   Transfert des variables graphiques A et B aux connecteurs suivants.

Cet exemple de création de bus conduit à la notion de macro commande.

3.4 Macro commande

Comme beaucoup de systèmes, MADMACS peut exécuter et mémoriser une séquence de commandes (macro commande ou macro) qui devient alors une nouvelle commande. En utilisant les déplacements contextuels et les variables graphiques, cette macro est indépendante des tailles et espacements entre objets. Elle peut alors s'exécuter dans tout contexte similaire à son contexte de définition. Les macros sont principalement utilisées pour les tâches répétitives. Par exemple, N exécutions successives de la macro suivante réalise un bus de N bits entre connecteurs, comme l'illustre la figure 4 :

```
(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)
```

[2] La création d'un rectangle nécessite deux points de référence : la marque courante et la position courante du curseur.

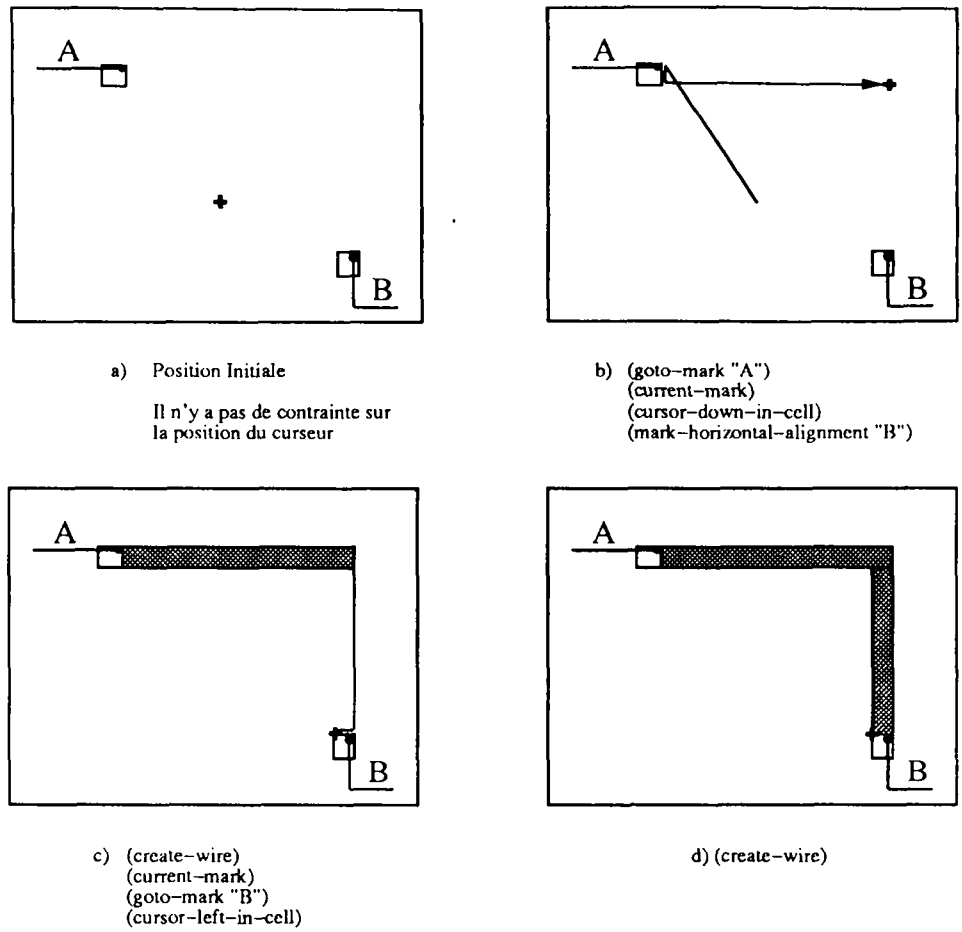


Figure 3: Création d'une connexion avec un angle droit

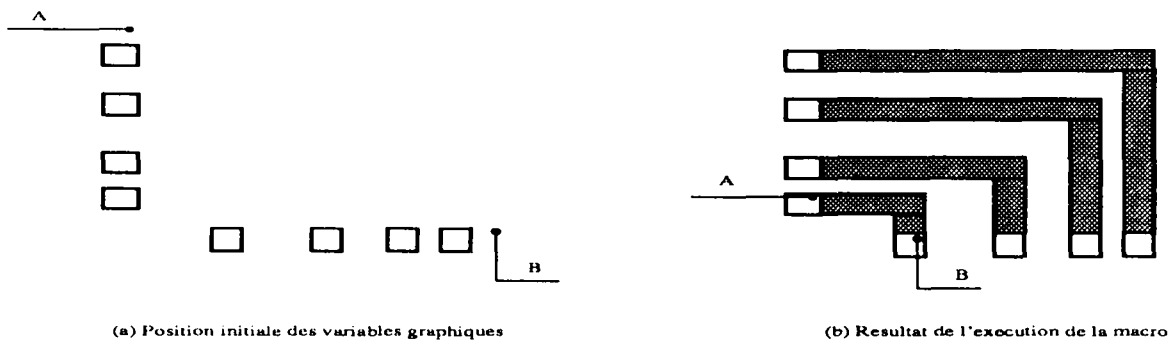


Figure 4: Exécutions multiples de la macro *(right-angle-connection)*.

où la commande *(make-a-right-angle-connection)* réalise une connexion avec angle droit.

Le mécanisme de macros est très puissant, mais ne permet pas de tester le contexte d'exécution. Par exemple, si la macro précédente est exécutée une cinquième fois, le résultat ne sera pas garanti. Pour tester ce contexte, la macro est sauvée sous une forme LISP équivalente et des instructions de contrôle lui sont ajoutées comme le montre la section suivante.

3.5 Interpréteur LISP

MADMACS fournit au concepteur une interface avec un langage interprété de haut niveau : LISP. Chaque commande MADMACS a une forme fonctionnelle dans le langage LISP. Quand l'interpréteur évalue une commande, il appelle MADMACS qui l'exécute et retourne un état d'exécution. Par exemple, (cursor-to-upper-cell) retourne 0 s'il y a un objet au dessus du curseur, 1 sinon.

Avec LISP, un concepteur peut définir des fonctions avec paramètres et contrôle. Par exemple, la fonction ci-dessous est une généralisation de la macro précédente où le paramètre *nbconnector* représente la largeur du bus. La fonction vérifie si les marques existent et avant de les déplacer, s'il y a un connecteur dans la direction donnée.

```
(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)
  )
)
```

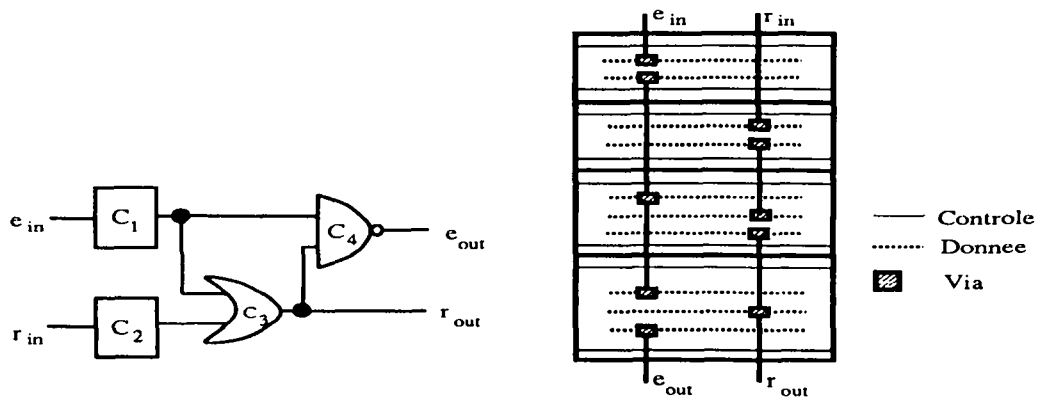
La même méthodologie, macro puis fonction LISP, est utilisée pour définir des générateurs. Un exemple est présenté par la suite.

4 Placement

4.1 Algorithme de placement

MadPlace prend en entrée une description de niveau portes logiques d'un processeur. A chaque élément de cette description correspond une cellule en librairie. Dans sa version actuelle, un processeur est obtenu par simple empilement de cellules. Les signaux de contrôle traversent horizontalement les cellules et ne nécessitent pas de routage. Par contre, les signaux de données doivent être routés verticalement par dessus les cellules.

La hauteur du processeur est égale à la somme des hauteurs des cellules. En revanche, sa largeur dépend en partie du nombre maximum d'interconnexions passant sur une cellule. Ce nombre est fonction de l'ordonnancement des cellules. Pour minimiser la surface,



(a) Processeur (b) Dessin correspondant

Figure 5: Exemple de placement.

il faut minimiser la coupe maximale du graphe ou de l'hypergraphe modélisant le processeur [7, 10, 17]. Dans le cas général, ce problème est NP-complet. Pour le résoudre, bon nombre d'algorithmes utilisent des heuristiques [10, 18]. Cependant, il est possible de mettre en œuvre un algorithme de décision bornant la coupe maximale d'un graphe [10]. L'algorithme de Gurari et Sudborough [7] a une complexité en $O(N^k)$, N étant le nombre de sommets du graphe et k la borne fixée. Il permet d'obtenir un sous-ensemble des ordonnancements linéaires du graphe ayant une coupe maximale $\leq k$. Cet algorithme est mis en œuvre dans MadPlace.

4.2 Exemple

Le circuit de la figure 5.a est utilisé pour illustrer la démarche de MadPlace. La solution obtenue a une coupe maximale de 2 pour l'ordonnancement $(C_1 - C_2 - C_3 - C_4)$. La figure 5.b montre l'assemblage correspondant. MadPlace donne en sortie une fonction LISP **assemble.mad**. Evaluée par le système MADMACS, elle assemble et route un processeur.

5 Un générateur de réseau linéaire

Le but de cette section est de montrer comment réaliser rapidement un générateur de réseaux. Considérons un réseau linéaire avec un plan de masse illustré par la figure 6. Les processeurs sont aboutés en colonnes, les signaux de contrôle sont distribués horizontalement de colonne en colonne en métal 1 et les signaux de données sont routés entre les colonnes en métal 2. Il est à noter que la dernière colonne peut avoir moins de processeurs que les autres.

Ce générateur utilise plusieurs paramètres : **nbcell** est le nombre total de processeurs, **nbcolumns** et **nbrows** sont respectivement le nombre de colonnes et de lignes. Au niveau réseau, un processeur est considéré comme une figure avec des connecteurs : **nbdata** et **nbcolumn** sont respectivement le nombre de connecteurs pour les signaux de données et pour les signaux de contrôle. Ce générateur utilise également quatre paramètres : largeur et espacement pour les couches de métal 1 et 2.

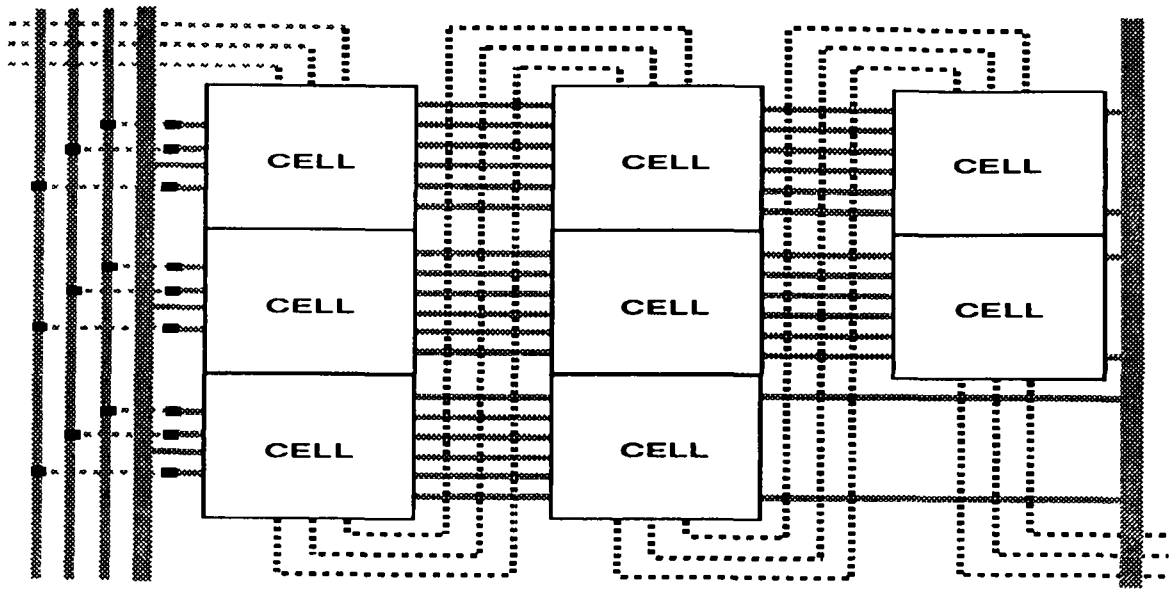


Figure 6: Plan de masse du réseau linéaire.

5.1 Contraintes sur les processeurs

Les signaux de données traversent verticalement les processeurs pour un aboutement vertical sans routage. Les fils de contrôle traversent les processeurs horizontalement pour une distribution à la colonne suivante. Le générateur considère un processeur comme une figure avec des connecteurs sur chaque côté: les verticaux pour le contrôle et les horizontaux pour les données.

5.2 Pavage de processeurs

Une fonction LISP est définie pour l'aboutement automatique d'une cellule au dessus d'une autre (c'est la généralisation de la séquence illustrée par la figure 2):

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

Cette fonction est étendue pour la création d'une colonne de **nbcell** processeurs:

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

Dans cette version, l'espacement entre deux colonnes de processeurs est calculé directement à partir des paramètres **nbdata**, largeur et espacement des couches de métal.

5.3 Routage des signaux de contrôle.

Le routage des signaux de contrôle s'effectue en deux étapes: distribution entre les colonnes et connexion aux lignes de signaux globales.

Distribution: Les contraintes sur les processeurs assurent qu'un connecteur et son opposé sur la colonne suivante aient la même ordonnée. La distribution est accomplie par une ligne horizontale entre un connecteur et la colonne suivante. La fonction suivante crée une telle connexion:

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

Comme montré précédemment, cette fonction peut être étendue pour réaliser successivement les connexions horizontales.

Connexion globale: Les lignes globales de contrôle sont créées à gauche du réseau (à l'exception de **vdd**). Le but est de créer une connexion horizontale entre un connecteur et la ligne globale associée. Cette correspondance est réalisée par l'identité entre le nom du connecteur et celui de la variable graphique associée à chaque ligne globale. La connexion est alors réalisée par un alignement horizontal entre un connecteur et la variable graphique associée.

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

En testant le nom du connecteur, cette fonction peut réaliser des connexions en métal 1 pour **gnd**, ou en métal 2 pour les autres. Une fonction similaire est utilisée pour les connexions à **vdd**.

5.4 Routage des signaux de données.

Le routage des signaux de données est réalisé en métal 2 et respecte les largeurs et espacements minimaux. Ce routage correspond à la création d'un bus avec angle(s) droit(s) entre deux ensembles de connecteurs. Pour les entrées et les sorties de données, les fonctions sont similaires à la macro *(right-angle-connection)* (voir figure 4). Pour le routage

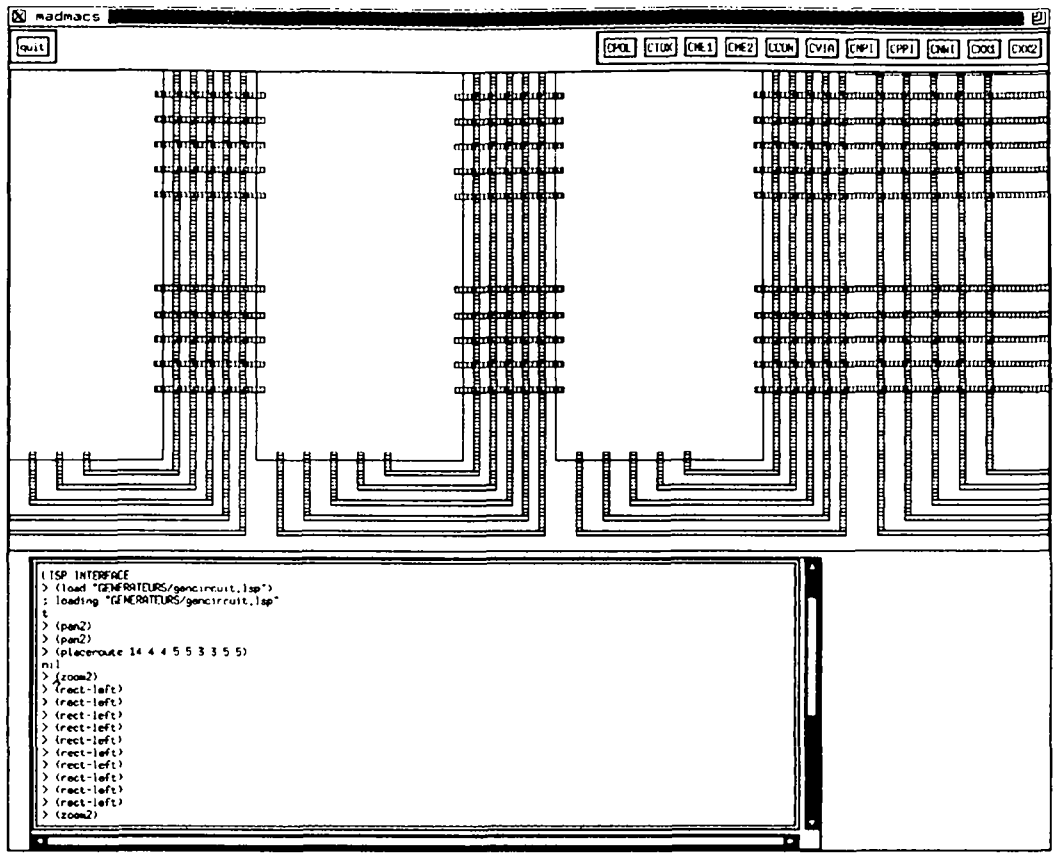


Figure 7: Interface MADMACS.

entre colonnes avec plusieurs angles droits, la fonction est plus complexe, car elle utilise plusieurs variables graphiques (une par angle droit).

Ces fonctions ainsi que les précédentes ont été construites suivant la même méthodologie. Tout d'abord, l'utilisateur crée interactivement une macro (en utilisant les déplacements contextuels et les variables graphiques). Quand cette macro est validée, elle est sauvée comme une fonction LISP et éditée pour la généraliser. Cette approche permet de définir rapidement des générateurs. En quelques heures, un concepteur peut créer une version initiale de ce générateur de réseaux linéaires.

6 Implémentation

MADMACS est développé en C++, un langage orienté objets. Les langages orientés objets [25] sont bien adaptés pour la manipulation d'objets comme ceux de MADMACS. C++ permet de développer facilement de nouvelles commandes et de les réutiliser. L'interface MADMACS (voir figure 7) est implantée sur le sytème X-Window :

- Les boutons du haut permettent de sélectionner une couche ou de quitter l'application en cours.
- La fenêtre graphique visualise les dessins des masques ainsi que le résultat des commandes de MADMACS.
- La fenêtre "texte" permet le dialogue avec l'interpréteur LISP.

Les commandes de MADMACS peuvent être entrées sous deux formes : fonctions LISP dans la fenêtre "texte", ou caractères associés aux commandes dans la fenêtre graphique. Cette dernière technique est celle d'éditeurs de textes comme Emacs et assure une très grande interactivité. Seul inconvénient, l'utilisateur doit mémoriser plusieurs touches. Cependant, ces touches associées aux commandes sont mnémoniques et donc faciles à retenir. L'expérience montre que l'apprentissage d'une telle interface est rapide. De plus, un concepteur utilise peu le système MADMACS pour concevoir une cellule, mais plutôt pour développer des générateurs. Il utilise l'interface graphique et surtout les macros pour construire interactivement un bloc et ainsi, mémorise automatiquement sa méthodologie de construction. Ensuite, il passe à une session d'édition de fonctions LISP pour enrichir cette première version obtenue interactivement.

7 Conclusion

Dans ce papier, nous avons décrit un sytème interactif de dessins de masques. Le concepteur peut définir des séquences de commandes indépendantes des tailles et espacements entre objets manipulés et les mémoriser sous forme de macro commandes. Ce sytème combine intelligemment les facilités du graphique et de la programmation. Une fois validée, une macro est sauvée comme une fonction LISP. Cette fonction peut être généralisée par ajout de contrôle et de paramètres.

Cette approche a plusieurs avantages. La définition de fonctions indépendantes des tailles et espacements est particulièrement importante. Le concepteur peut ainsi construire des librairies de fonctions et les utiliser pour définir des générateurs. Par ailleurs, il est souvent difficile de donner directement une version textuelle de la construction d'un bloc. Grâce au mécanisme interactif de macros, le concepteur peut mémoriser facilement cette construction et ainsi capturer sa méthodologie. Puis, en éditant cette première version obtenue interactivement, il peut la généraliser. En suivant cette méthodologie, notre expérience a montré qu'un concepteur développe rapidement de nouveaux générateurs.

References

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

[2] JM. Berge, L. O. Donzelle, J. Rouillard & D. Rouquier. LOF. Note Technique, CNET-FRANCE, 1985.

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

[4] C. Dezan, E. Gautrin, H. Le Verge & P. Quinton. Synthesis of systolic arrays by equation transformations. Dans *ASAP 91*, Barcelone, Espagne, Septembre 1991.

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

[6] E. Gautrin & L. Perraudeau. MADMACS: a tool for the layout of regular arrays. Dans *IFIP*, Mars 1992.

[7] E. M. Gurari & I. H. Sudborough. Improved Dynamic Programming Algorithms for Bandwidth Minimzation And The MinCut Linear Arrangement Problem. Dans *Journal of Algorithms*, 5, pages 531–546, 1984.

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

[9] T. Lengauer. The Combinatorial Complexity of Layout Problem. Dans *Physical Design Automation of VLSI Systems-Chp.10*, édité par Bryan T. PREAS & Michael J. Lorenzetti, pages 461–497, 1988.

[10] T. Lengauer. Combinatorial Algorithms for Integrated Circuit Layout. Edité par B. G. Teubner and John Wiley & Sons, 1990.

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

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

[13] F. S. Makedon, C. H. Papadimitriou & I. H. Sudborough. Topological Bandwidth. Dans *SIAM J. Alg. Disc. Meth.*, vol. 6, $N°3$, pages 418–444, Juillet 1985.

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

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

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

[17] Z. Miller & I. H. Sudborough. A Polynomial Algorithm for Recognizing bounded Cutwidth in hypergraphs. Dans *Math. Systems Theory*, 24, Springer-Verlag, pages 11–40, 1991.

[18] B. T. Preas & P. G. Karger. Placement, Assignement and Floorplanning. Dans *Physical Design Automation of VLSI Systems-Chp.4*, édité par Bryan T. PREAS & Michael J. Lorenzetti, pages 461–497, 1988.

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

[20] H. Le Verge and C. Mauras & P. Quinton. The ALPHA Language and its Use for the Design of Systolic Arrays. Dans *Journal of VLSI Signal Processing*, Volume 3, pages 173-182, 1991.

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

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

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

[24] VLSI Technology Inc. *Datapath Compiler.* Rapport technique, VLSI Technology Inc., San Jose, CA, USA, Juin 1989.

[25] W. Wolf. Object-Oriented Programming for CAD. Dans *IEEE Design and Test of Computers*, pages 35-42, Mars 1991.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 642 ARCHE : UN LANGAGE PARALLELE A OBJETS FORTEMENT TYPES
Marc BENVENISTE, Valérie ISSARNY
Mars 1992, 132 pages.

PI 643 CARTESIAN AND SATISTICAL APPROACHES OF THE SATISFIABILITY PROBLEM
Israël-César LERMAN
Mars 1992, 58 pages.

PI 644 PRIME MEMORY SYSTEMS DO NOT REQUIRE EUCLIDEAN DIVISION BY A PRIME NUMBER
André SEZNEC, Yvon JEGOU, Jacques LENFANT
Mars 1992, 10 pages.

PI 645 SKEWED-ASSOCIATIVE CACHES
André SEZNEC, François BODIN
Mars 1992, 20 pages.

PI 646 INTERLEAVED PARALLEL SCHEMES : IMPROVING MEMORY THROUGHPUT ON SUPERCOMPUTERS
André SEZNEC, Jacques LENFANT
Mars 1992, 14 pages.

PI 647 COMMUNICATING PROCESSES AND FAULT TOLERANCE : A SHARED MEMORY MULTIPROCESSOR EXPERIENCE
Michel BANATRE, Maurice JEGADO, Philippe JOUBERT, Christine MORIN
Mars 1992, 40 pages.

PI 648 SET-THEORETIC GRAPH REWRITING
Jean-Claude RAOULT, Frédéric VOISIN
Mars 1992, 18 pages.

PI 649 UNE STRUCTURE D'INFORMATION POUR LES ALGORITHMES D'EXCLUSION MUTUELLE FONDES SUR UNE ARBORESCENCE
Jean-Michel HELARY, Achour MOSTEFAOUI, Michel RAYNAL
Mars 1992, 18 pages.

PI 650 BLOCK-ARNOLDI AND DAVIDSON METHODS FOR UNSYMMETRIC LARGE EIGENVALUE PROBLEMS
Miloud SADKANE
Avril 1992, 24 pages.

PI 651 COMPILING SEQUENTIAL PROGRAMS FOR DISTRIBUTED MEMORY PARALLEL COMPUTERS WITH PANDORE II
Françoise ANDRE, Olivier CHERON, Jean-Louis PAZAT
Avril 1992, 18 pages.

PI 652 CHARACTERIZING THE BEHAVIOR OF SPARSE ALGORITHMS ON CACHES
Olivier TEMAM, William JALBY
Avril 1992, 20 pages.

PI 653 MADMACS : UN OUTIL DE PLACEMENT ET ROUTAGE POUR LE DESSIN DE MASQUES DE RESEAUX REGULIERS
Eric GAUTRIN, Laurent PERRAUDEAU, Oumarou SIE
Avril 1992, 16 pages.

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

ISSN 0249 - 6399