

# MADEO

## *Une chaine de CAO générique pour le reconfigurable*

Loïc LAGADEC

Équipe Architectures&Systèmes

UBO – France

<http://as.univ-brest.fr>

## Présentation rapide

- Madeo est une chaîne de CAO pour les architectures reconfigurables
- Objectifs: Favoriser la réutilisation d'outils, d'applications (portabilité), dimensionnement d'architectures pour un jeu d'applications, choix ouvert d'arithmétiques
- Moyens: Approche modulaire; disjonction outils/architectures et applications/plages de données
- Durée: Début des travaux en 95/96

## Plan de l'exposé

- Introduction
  - Rappel sur les Architectures Reconfigurables
  - Rappel sur le codage VHDL
- MADEO
  - Couche haute
  - Couche basse
- Conclusion

## Usages des technologies reconfigurables

### Deux usages principaux pour les architectures reconfigurables

- *Remplacement d'ASICs*
  - *Prototypage* → *Rapidité, faible coût*
  - « *Petites* » séries → *De moins en moins petites*
  - *Support multitâche* → *Reconfiguration*
- *Besoins évolutifs*
  - *cadre embarqué*
  - *Accélérateur matériel*

## Exécution reconfigurable

Données

configuration

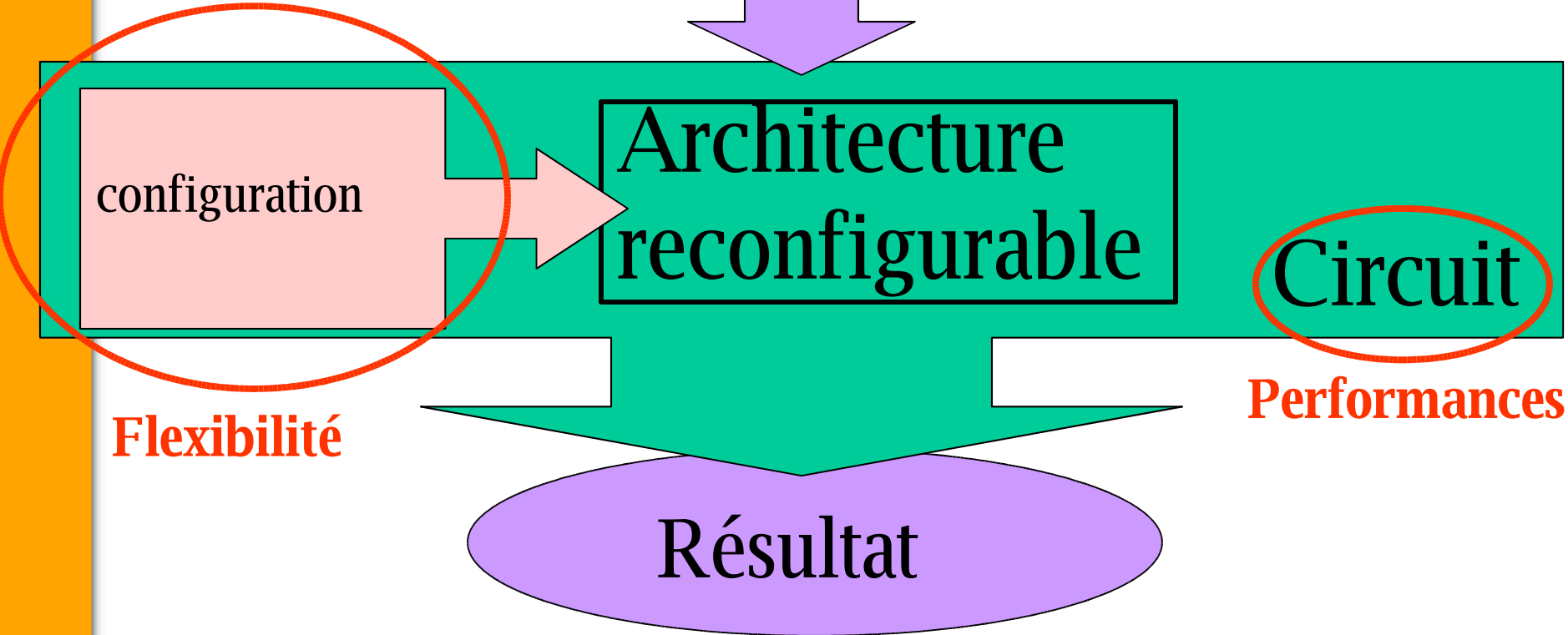
Architecture  
reconfigurable

Circuit

Flexibilité

Performances

Résultat



## Les évolutions matérielles sont rapides

- Nouvelles caractéristiques (capteurs, SOC, ...)
- Nouvelles fonctionnalités (configuration dynamique)
- Nouvel usage (Intégration système, partage, ...)
- Nouvelles métriques (basse conso,...)
- Séries expérimentales, ...
- Nouvelles technologies
  
- Quantité de silicium et couts de réalisation augmentent

## Les limitations

D'une part les ressources augmentent mais sont mal hiérarchisées  
(problème d'architectures)

D'autre part la programmation s'effectue en aveugle par rapport à  
l'architecture (problème d'outils, structuration en couches)

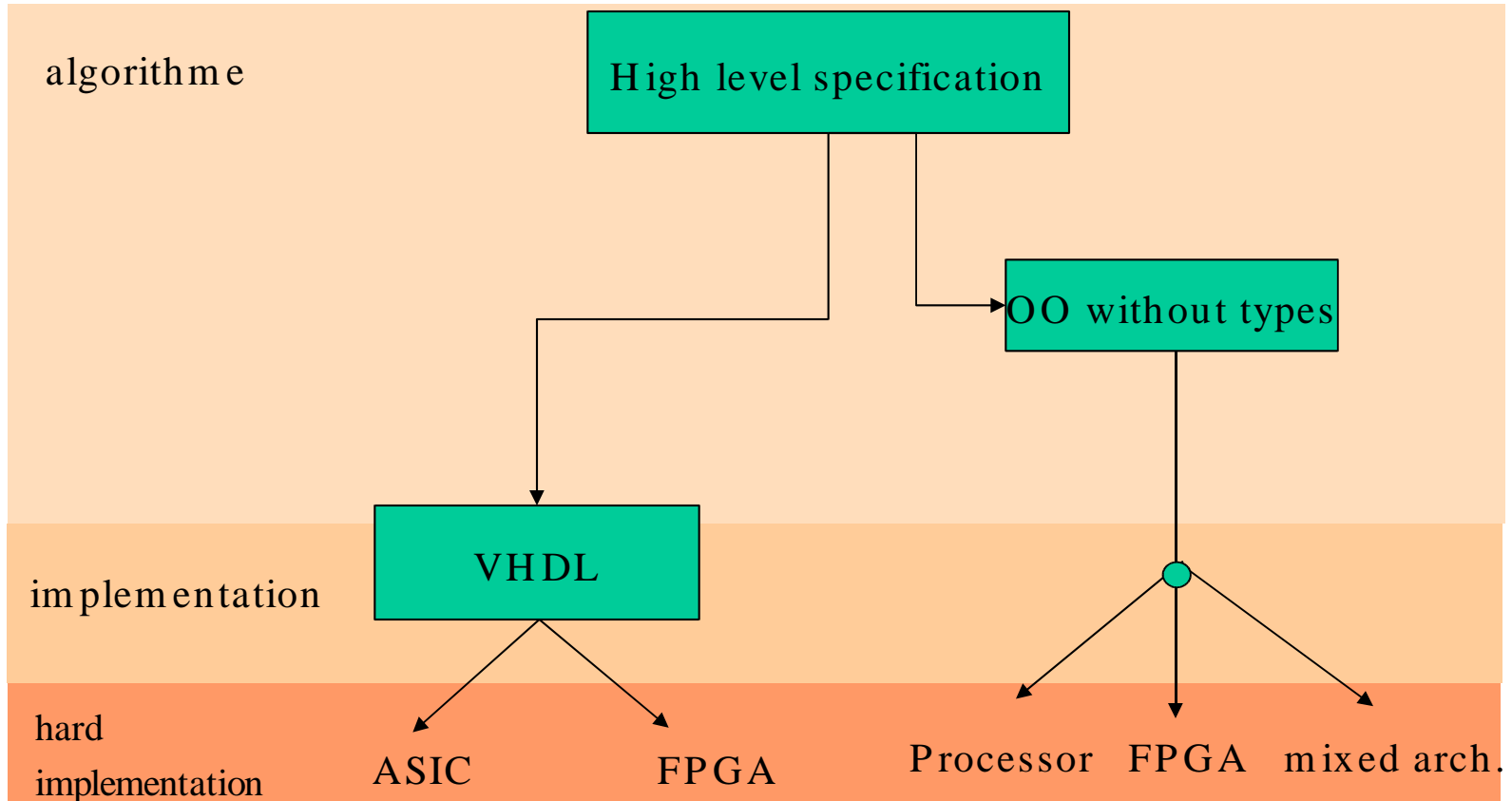
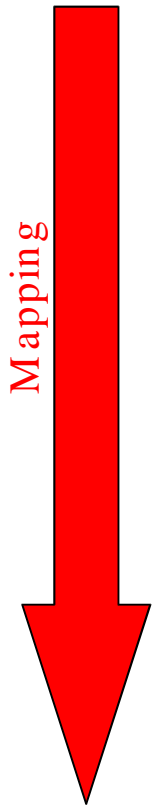
Enfin l'expression des traitements à implanter est héritée du VLSI  
(problème de niveau conceptuel en entrée)

Avons nous compris la richesse de ces architectures?

# VHDL vs Programmation symbolique

- Préserver la sémantique de haut niveau (indépendante du contexte)
- Auto adaptation au contexte (valeurs + cible)

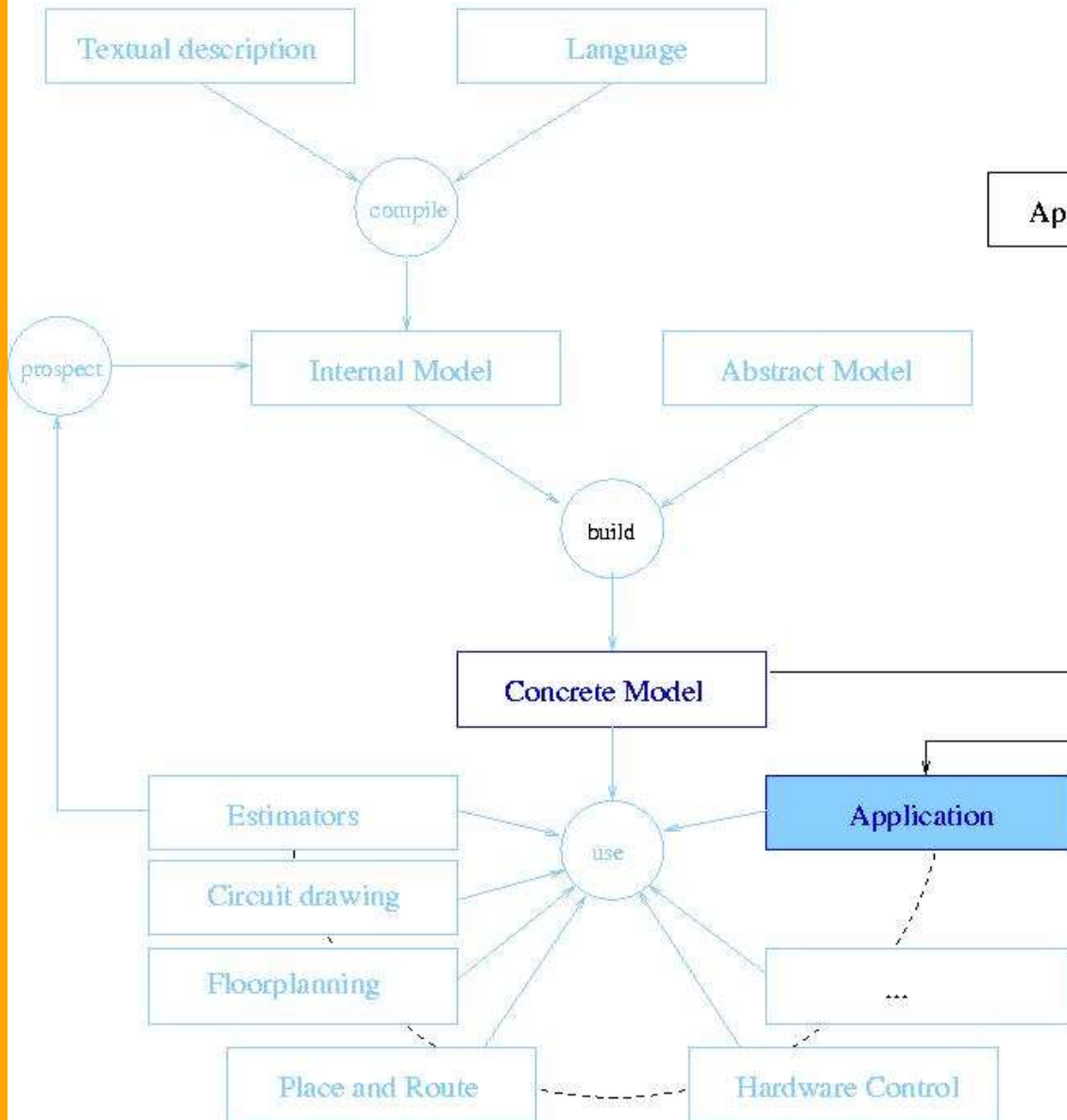
Haut niveau  
d'abstraction



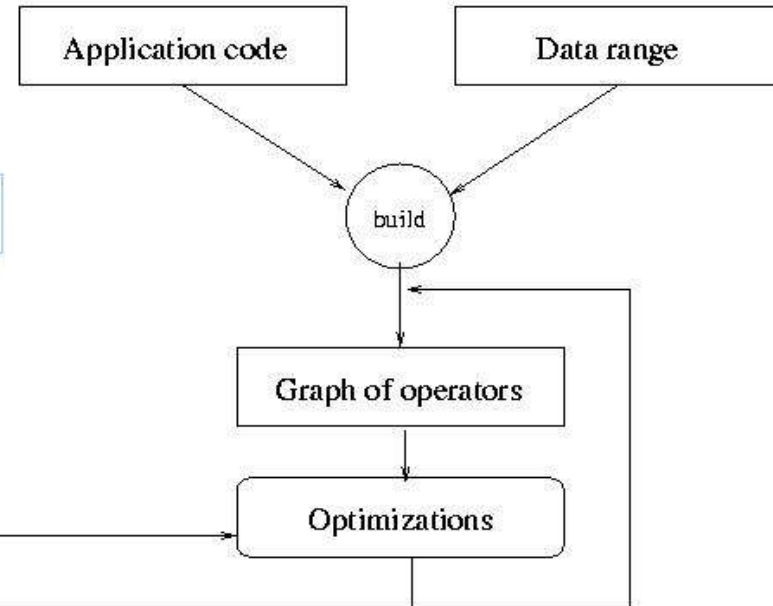
Bas niveau

# Flot Madeo Global

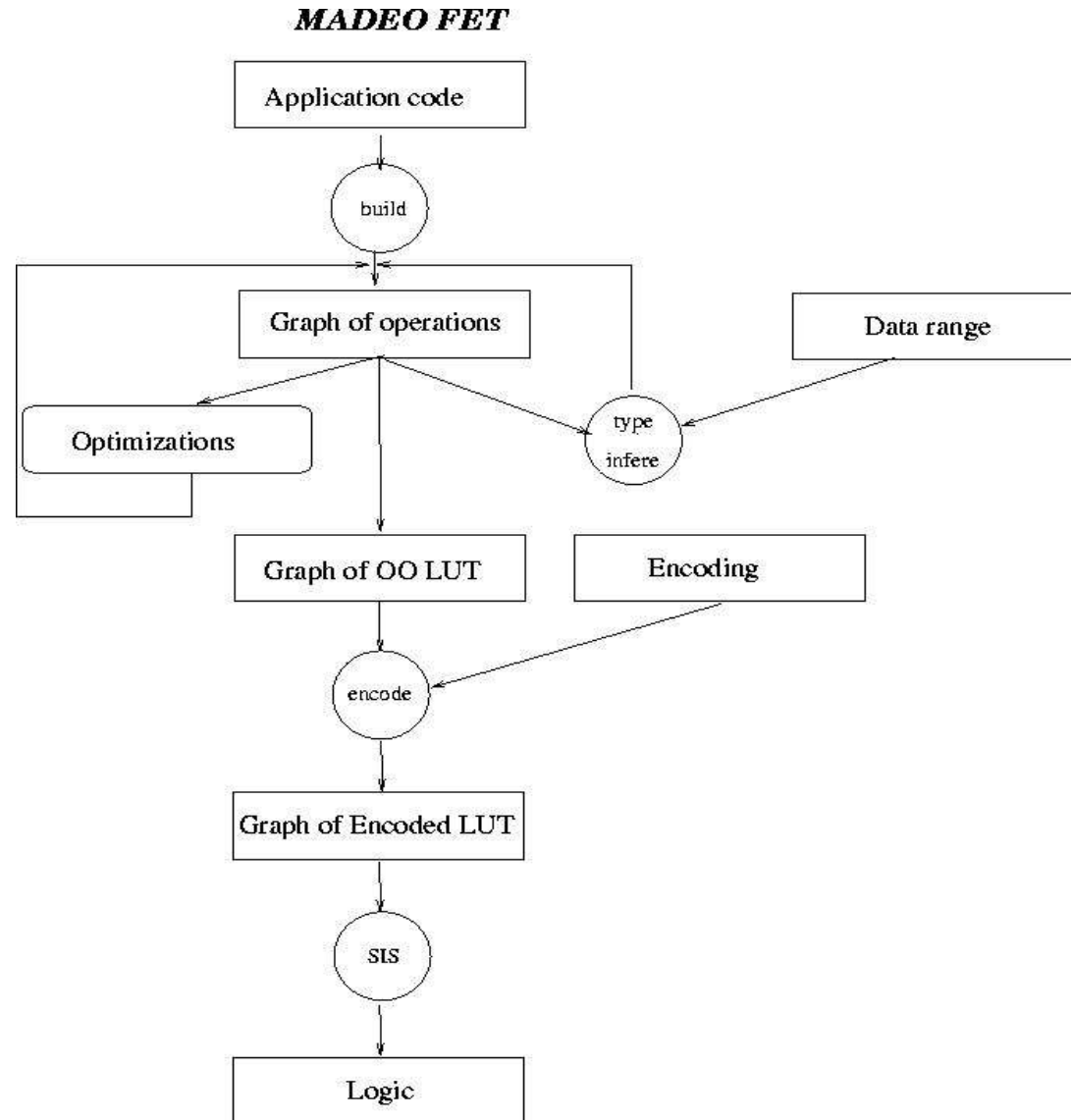
## MADEO BET



## MADEO FET



## Flot de conception



## Le code

- Le code est symbolique
  - Fonctionnel
  - Non typé
- Le code est situé dans une classe
  - Les opérations portant sur des objets utilisent une évaluation classique
  - Certaines opérations sont structurantes (appels fonctionnels)
- Le typage est extérieur
- Le résultat est contextuel

## Compilation du code de haut niveau

- La compilation respecte l'écriture du code puis des optimisations sont réalisées
  - Chaque opération produit un noeud
  - Les noeuds peuvent être hiérarchiques (évaluation paresseuse)
- La hiérarchie peut être manipulée
  - Suivant des annotations
  - A la main
  - En fonction du contexte

## Optimisations

- Les optimisations sont classiques
  - Factorisation
  - Suppression de code mort
  - Propagation de constantes
  - ...
- On ajoute essentiellement l'inférence de type

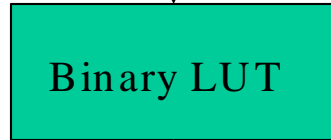
## Inférence de types

- L'inférence de type propage les types
  - d'entrée
  - calculés en sortie des opérateursd'opérations en opérations
- Un type est une énumération de valeurs possibles
- Seuls sont considérés les n-uplets possibles de sorte que la logique produite est minimale
- Un autre bénéfice est de différer la considération de la sémantique des opérateurs  
exemple: changer l'arithmétique

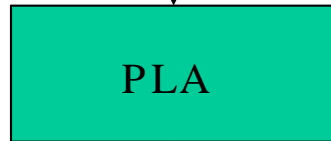
## Synthèse logique



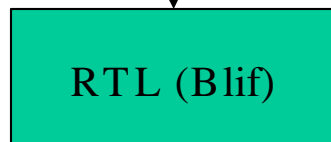
- Chaque noeud connait sa “table de vérité”



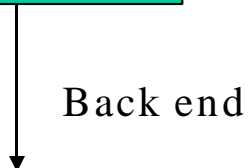
- Les valeurs peuvent être encodées



- On obtient un PLA à minimiser (expresso)



- La synthèse logique repose sur SIS (UC Berkeley)



## Exemple: Multiplication flottante

- Chaque nombre est représenté par trois valeurs
  - **sign** (0,1)
  - **exponent** (from -x to +y using  $\log_2(x+y+1)$  bits)
  - **significand** (from 1 to  $2 - \epsilon$  using  $\text{abs}(\log_2(\epsilon))$  bits)
- $\text{sign} = \text{signA} + \text{signB} \text{ modulo } 2$
- $\text{significand} = \text{trunc}(\text{significandA} * \text{significandB})$
- $\text{exponent} = \text{exponentA} + \text{exponentB} + \text{shift}$

## Exemple: Multiplication flottante

| sign exp significand normalize |

sign := self computeSignFor: signA and: signB.

significand := self computeSignificandFor: significandA

and: significandB.

exp := self computeExponentFor: exponentA and: exponentB.

normalize := self normalizeSignificand: significand.

^Array

with: sign

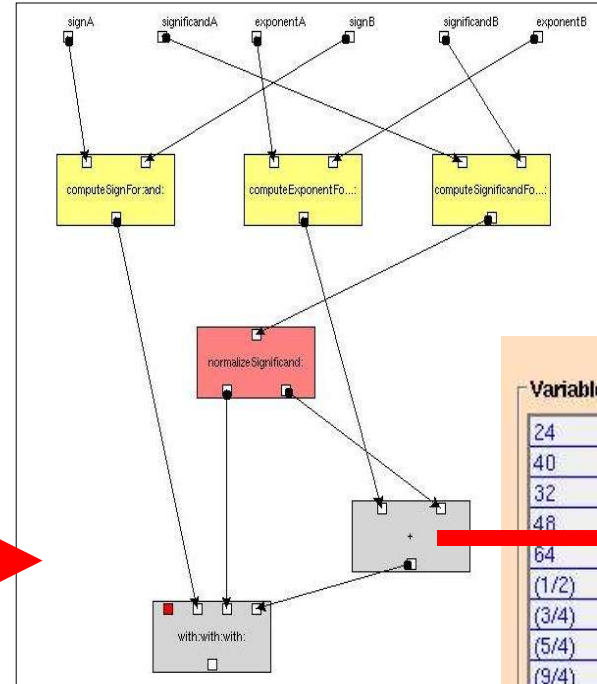
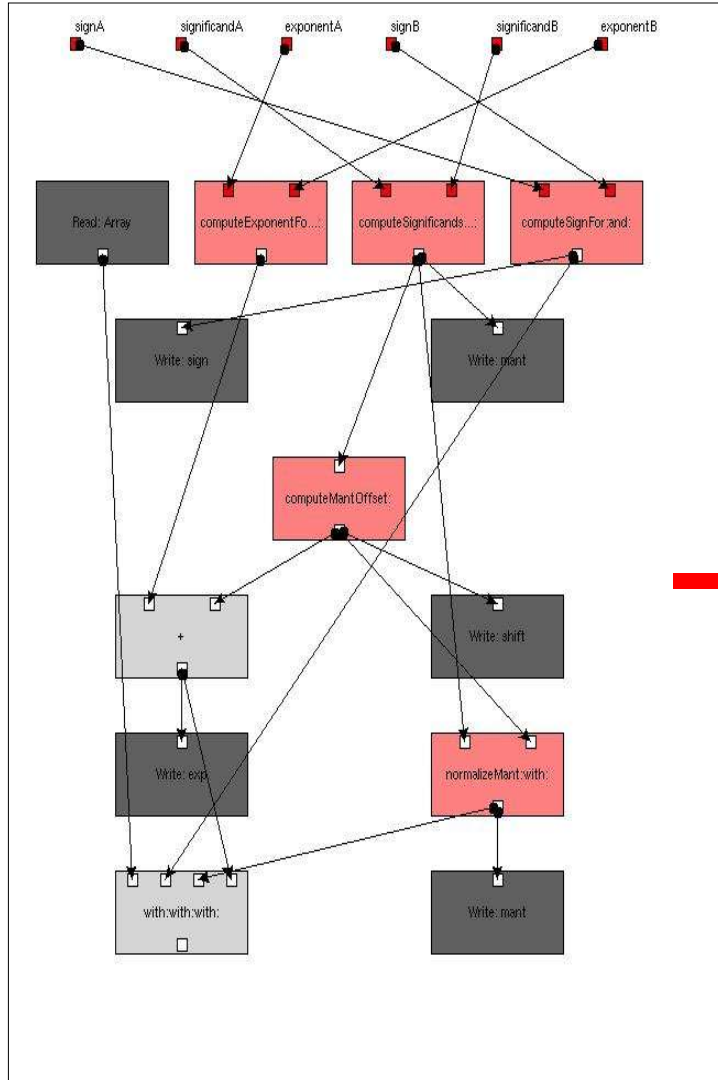
with: (normalize at: 1)

with: exp + (normalize at: 2)

Appels fonctionnels

Resultat tabulé

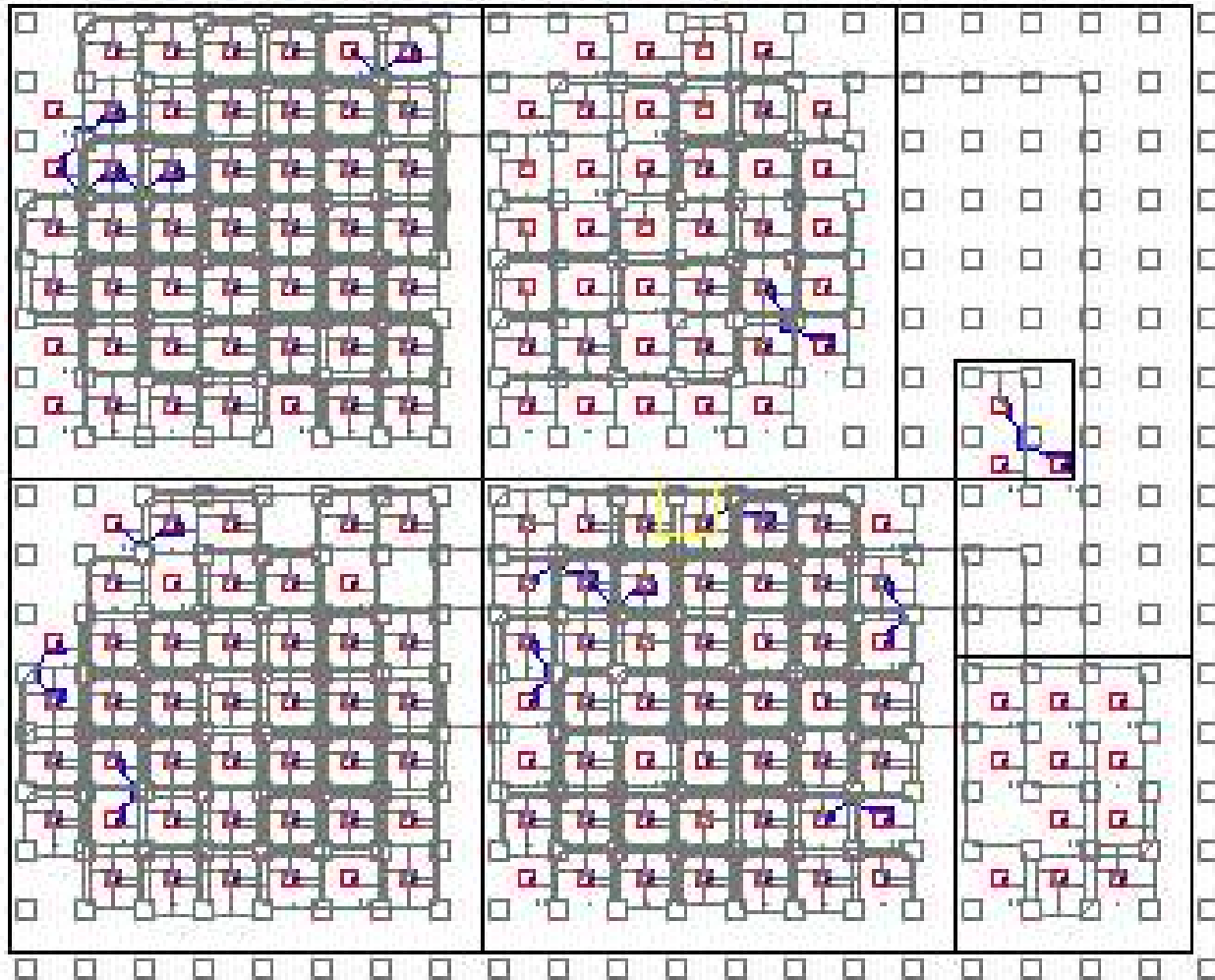
# Compilation / optimisation



**Variables - Results**

|         |   |            |
|---------|---|------------|
| 24      | 0 | #(24)      |
| 40      | 0 | #(40)      |
| 32      | 0 | #(32)      |
| 48      | 0 | #(48)      |
| 64      | 0 | #(64)      |
| (1/2)   | 1 | #((3/2))   |
| (3/4)   | 1 | #((7/4))   |
| (5/4)   | 1 | #((9/4))   |
| (9/4)   | 1 | #((13/4))  |
| (17/4)  | 1 | #((21/4))  |
| (33/4)  | 1 | #((37/4))  |
| (65/4)  | 1 | #((69/4))  |
| (129/4) | 1 | #((133/4)) |
| 1       | 1 | #(2)       |
| (3/2)   | 1 | #((5/2))   |
| (5/2)   | 1 | #((7/2))   |
| (9/2)   | 1 | #((11/2))  |
| (17/2)  | 1 | #((19/2))  |
| (33/2)  | 1 | #((35/2))  |

## Layout (Floorplanning + placement routage)



## Rappel

- *L'objectif est de permettre une portabilité transparente*
- *Pour cela les outils de bas niveaux doivent*
  - *Etre à meme d'implémenter les circuits optimisés*
  - *Fournir un retour d'information aux outils précédents*
  - *Offrir une API stable*
- *Les outils disponibles permettent ils cela?*

NON



- *appels à des bibliothèques d'opérateurs*
- *comportement imprévisible*
- *évolution incontrôlable*

ALORS



- *créer des outils de bas niveau adaptés*

## Contraintes sur les outils de back end

- *Il faut:*

|   |                        |
|---|------------------------|
| <ul style="list-style-type: none"><li>– <i>Production rapide d'outils</i></li><li>– <i>Disponibilité d'outils pour la prospection</i></li></ul>             | <i>Besoins</i>         |
| <ul style="list-style-type: none"><li>– <i>Les mêmes outils pour toute architecture</i></li><li>– <i>Outils adaptables (ex: nanotechnologies)</i></li></ul> | <i>Propriétés</i>      |
| <ul style="list-style-type: none"><li>– <i>Pleine exploitation du matériel par les outils</i></li><li>– <i>Support pour intégration système</i></li></ul>   | <i>Fonctionnalités</i> |

## *Solution proposée*

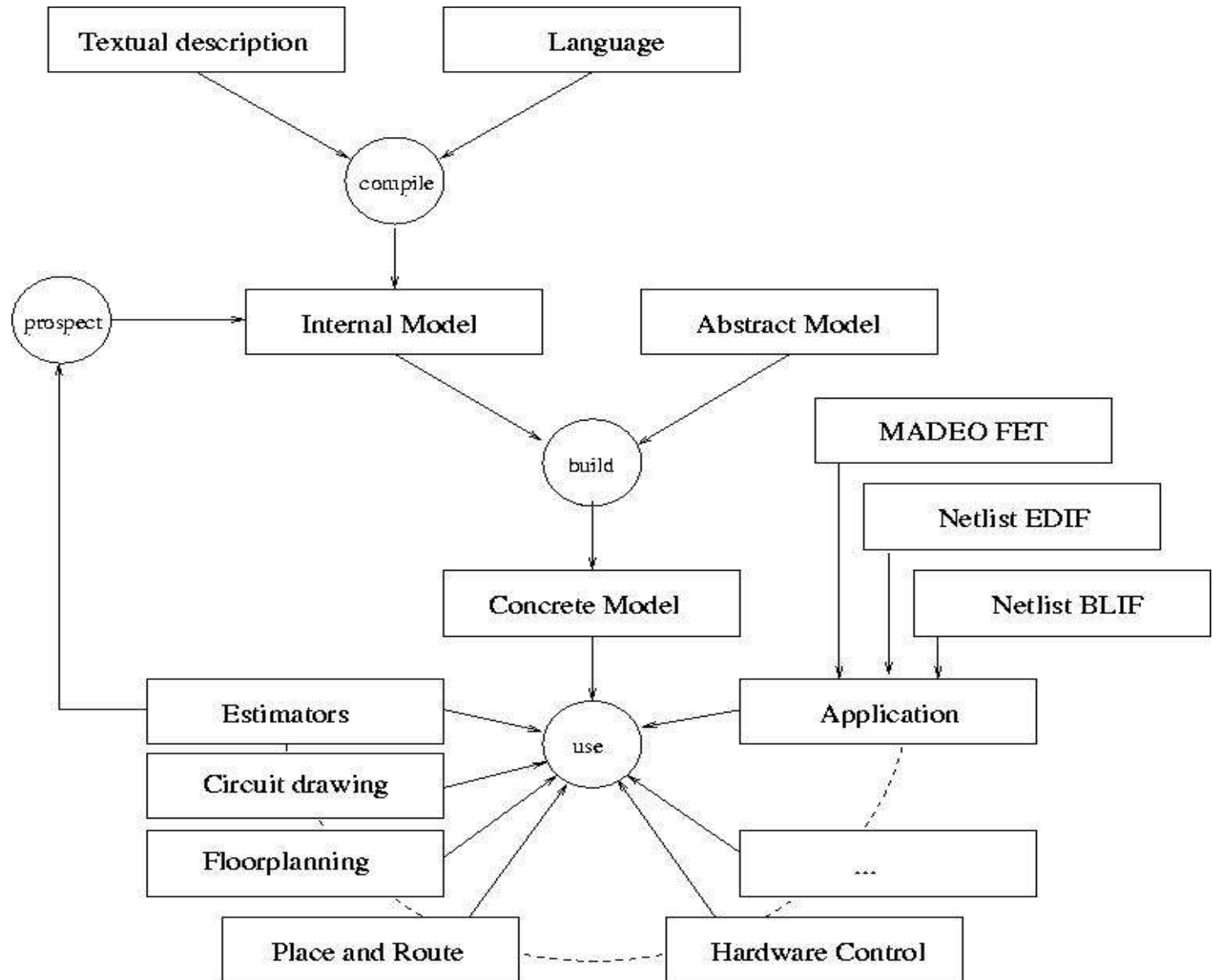
- *Transposer une solution qui a fait ses preuves*
- *Suivre les recommandations de la méthodologie objet*
  - *Abstraction des traitements*
  - *Modélisation du support*
- *Disjonction entre les traitements à réaliser et le support manipulé*

 *Madeo-Bet*

## Madeo-bet

- *Définir une représentation unifiée des différentes architectures possibles*
- *Définir un ensemble d'outils agissant sur cette représentation*
- *Il faut garantir*
  - *Que toute architecture est représentable de la sorte (possibilité d'extension)*
  - *Que la description de l'architecture reste un procédé léger et rapide (compilation)*
  - *Que les outils sont efficaces (algorithmes reconnus) et répondent aux besoins pré-cités (tolérance aux pannes, ...)*
  - *Que les outils sont facilement commandables, inter-opérables, et le cas échéant, remplaçables*

## Flot Madeo-Bet



## Le langage

- Un langage de description de la cible
  - Décrit les différents éléments de l'architecture
  - Décrit la représentation de l'architecture
- Les éléments sont de deux types:
  - Structuration hiérarchique
    - Pavage
    - Composite
  - Élément atomique
    - Fonction
    - Canaux
    - Multiplexeurs ...
  - + Référence à des objets déjà décrits

## Les outils

- Les outils sont définis de manière génériques
  - Visualisation / Commande
  - P&R (Lee, PathFinder)
  - Floorplanning / Tracé régulier
  - Diverses métriques
  - Interfaçage Matériel
  - Simulation à différents niveaux
  - Prospection architecturale
- Les outils se spécialisent en fonction du modèle d'architecture
- Les outils constituent un cadre ouvert

# Simulation des circuits

**LPPGAArray Editor**

File Modules Synthesis Representation Expert Mode Help

selection size : 0  
8@8

4

0%

t28  
 t38  
 t15  
 t7  
 t19  
 multiplier

Paste  
 Merge  
 Clear Instance  
 Remove from the list  
 Generate bitstream  
 Simulate  
 Print on file

**SCNode Simulator**

Input values

|       |   |
|-------|---|
| signA | 1 |
| signB | 0 |

Output values

|     |   |
|-----|---|
| t28 | 1 |
|-----|---|

Evaluate  
Clock  
Read Inputs  
Inspect

**SCNode Simulator**

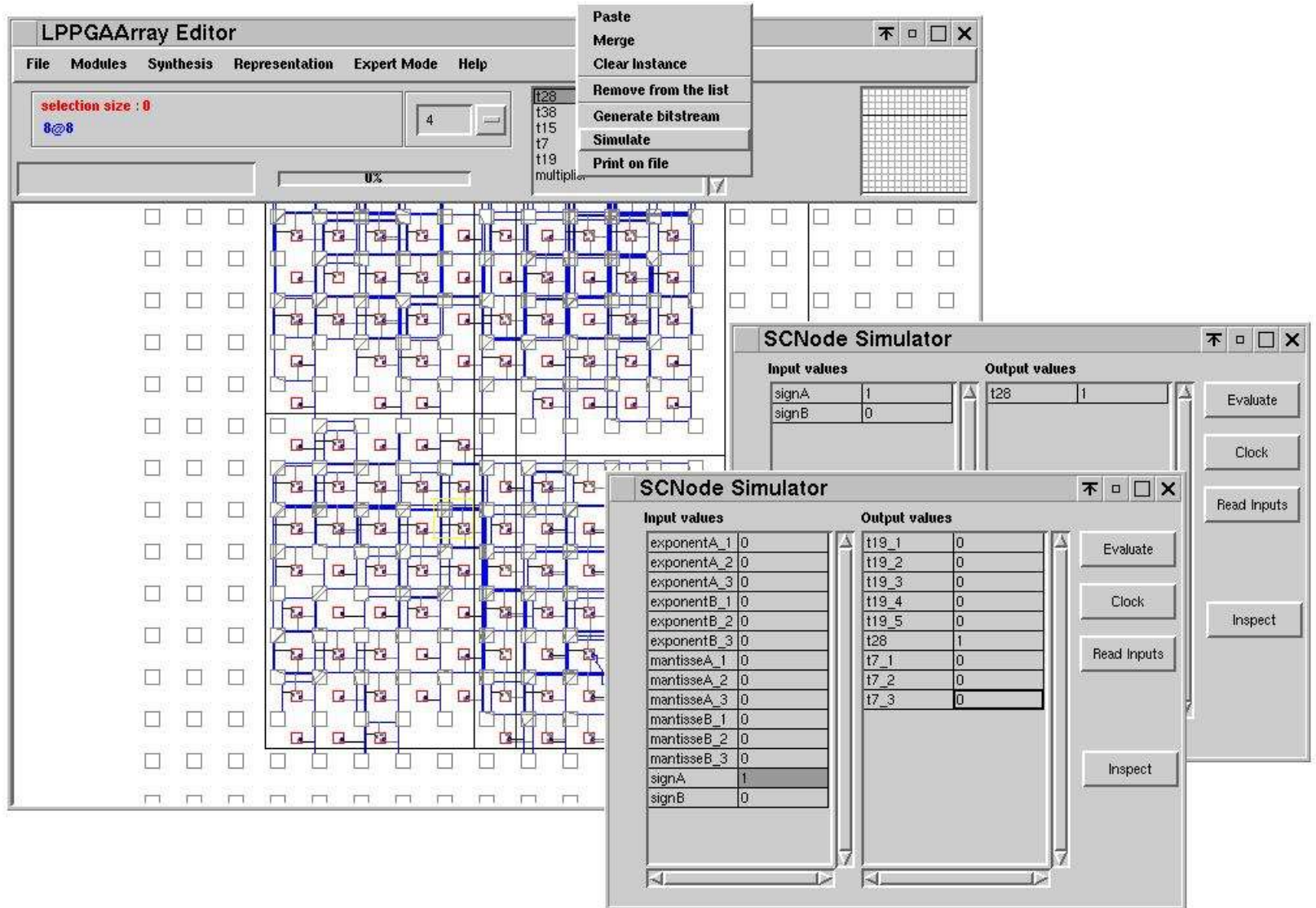
Input values

|             |   |
|-------------|---|
| exponentA_1 | 0 |
| exponentA_2 | 0 |
| exponentA_3 | 0 |
| exponentB_1 | 0 |
| exponentB_2 | 0 |
| exponentB_3 | 0 |
| mantisseA_1 | 0 |
| mantisseA_2 | 0 |
| mantisseA_3 | 0 |
| mantisseB_1 | 0 |
| mantisseB_2 | 0 |
| mantisseB_3 | 0 |
| signA       | 1 |
| signB       | 0 |

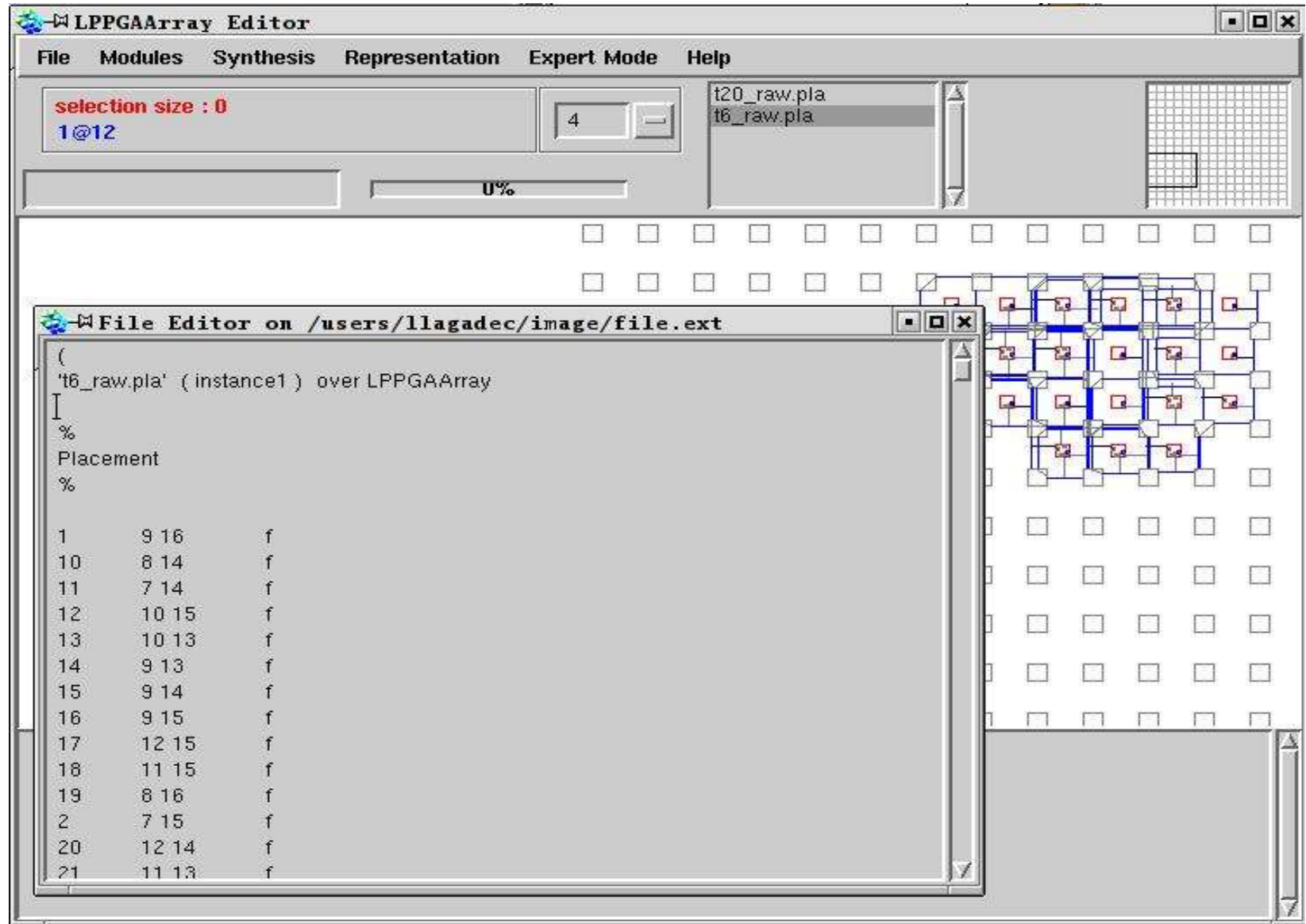
Output values

|       |   |
|-------|---|
| t19_1 | 0 |
| t19_2 | 0 |
| t19_3 | 0 |
| t19_4 | 0 |
| t19_5 | 0 |
| t28   | 1 |
| t7_1  | 0 |
| t7_2  | 0 |
| t7_3  | 0 |

Evaluate  
Clock  
Read Inputs  
Inspect



## Interface avec des outils tiers

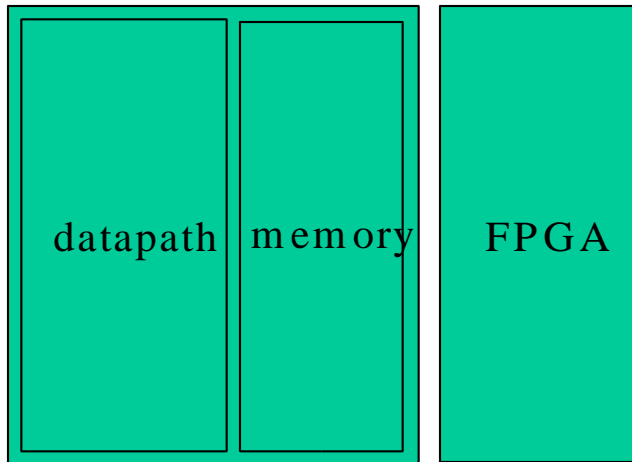


## Virtex 1 / Jbits / Celoxica Board

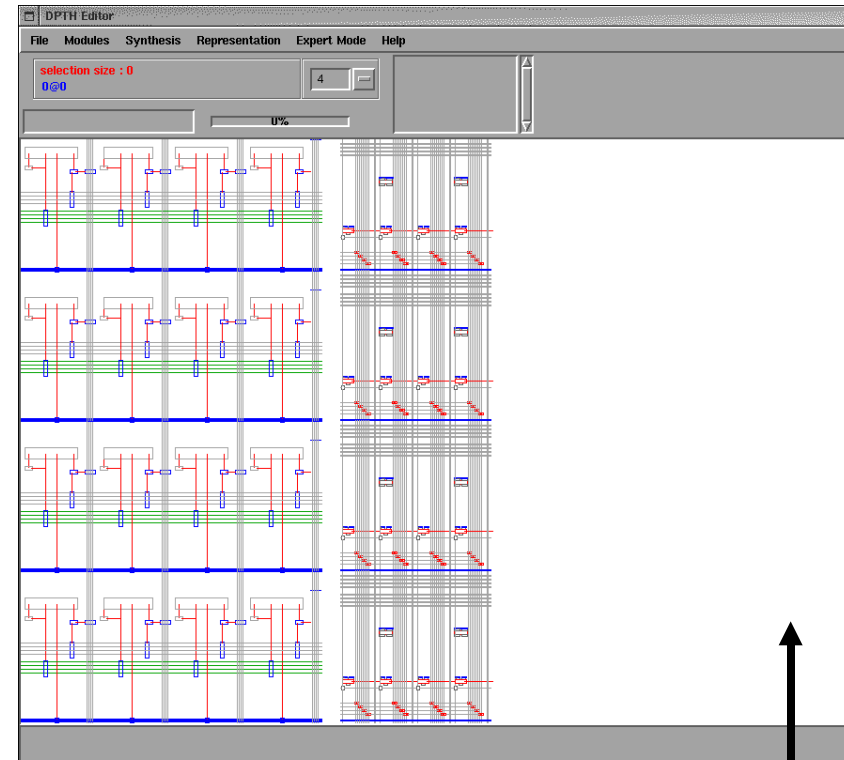
The image displays two software windows. The left window, 'Virtex Editor', shows a grid-based layout of a Virtex 1 device with various components and routing paths. The right window, 'BoardScope', shows a 'Routing Density View' with a color-coded grid and a detailed circuit diagram for 'Slice 1' and 'Slice 0'. A large black arrow labeled 'JBits' points from the Virtex Editor to the BoardScope window. Another black arrow points from the BoardScope window to a box labeled 'Hardware'. A third arrow points from the BoardScope window to a 'Properties' window on the right, which lists details for a component named 'SAMOSO', including its type, hierarchy, and input ports.

- Sous réserve de disponibilité d'une l'API, des outils tiers peuvent être exploités
- Exemple, programmation de Virtex en appui sur JBits

## Architecture hétérogène



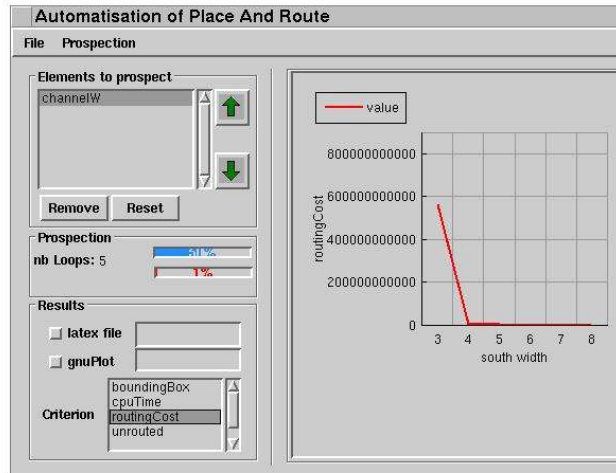
- Architecture hétérogène
- FPGA + reconfigurable datapath



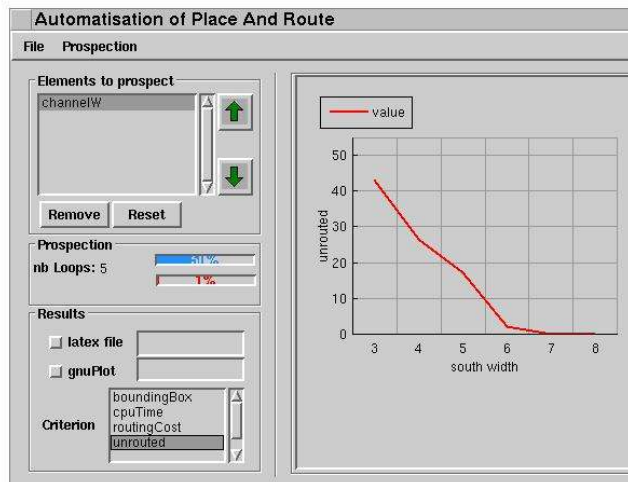
- Nouvelle description dans le langage
- Nouvelle analyse de code (extraction du controle)
- Nouveaux algorithmes de placement (synthèse d'architecture,...)

Specialisation des outils

## Prospection architecturale



- Il est possible de faire varier les architectures
  - De façon programmée
  - Par annotation dans le programme



- On déclenche des P&R et on collecte les résultats

## Conclusion

Madeo garantit :

- Fort degré de réutilisation des outils
- Temps et couts de développement réduits
  
- Une portabilité du code source des applications (y compris encapsulant une sémantique métier)
- Une optimisation automatique des circuits
  
- La prise en charge d'architectures nouvelles / de technologies émergentes (moléculaires, QCA, ...)
- La définition d'arithmétiques arbitraires (RAID / reed-solomon GF16, Turbo Code en GF128)

Madeo est un cadre de développement ouvert

*Merci de votre attention*

*<http://as.univ-brest.fr>*