Slide 1

Architectures reconfigurables et environnements logiciels

Bernard Pottier

Architectures et Systèmes, UBO.

pottier@univ-brest.fr

École CNRS Systèmes enfouis
Seix, 22 Novembre 2000

Slide 2

Contents

Slide 3

1 Présentation du domaine

1.1 Reconfigurable : une définition

- Ensemble de fonctionalités dont le comportement peut être changé individuellement,

Modèle de référence: les FPGAs, où l'on va pouvoir changer des connexions, des comportements logiques, des aiguillages...

Autres références: outils logiciels objets, réseaux...

Slide 4

1.2 Fondamentalement

- Interprétation d'une architecture B sur une autre architecture A,
- Aujourd'hui le silicium, demain ?...
- Les descriptions de configurations sont très complexes,
- Le séquencement temporel est une dimension du problème.

Slide 5

1.3 Applications

- Le marché des composants reconfigurables a un taux d'expansion important avec des applications dans le domaine des communications, du prototypage, du test...
- Ils apportent flexibilité et vitesse de production et permettent le développement d'applications pour des marchés de taille moyenne.

Slide 6

1.4 Caractéristiques et évolutions

- Actuellement 75000/50000 éléments logiques chez Xilinx/Altera
- Fréquence système de l'ordre de 200Mhz
- Architecture de routage et modularité pour faciliter l'usage des plus gros circuits.

Développement d'applications: principalement langages de description matériel, modules IP.

Slide 7

1.5 Reconfigurable et ULSI

- Nécessité de trouver des plateformes généralistes permettant l'intégration rapide de grandes applications sans passage en fonderie: la taille du marché grandit avec la taille de l'application et ceci a des limites.
- Tout logiciel sur multiprocesseur intégré? problèmes de grain logique, de concurrence et de temps réel,
- Tout reconfigurable? pas avec les technologies actuelles: place, vitesse, ...

Multiprocesseurs compacts (SMT)? distribués et reconfigurables? Quelle place pour la logique reconfigurable?

Slide 8

1.6 Quelques projets de recherche

- Accélérateurs 'chemins de données' (PipeRench, Chimaera)
- Approche système (Score, travaux sur le 6200)
- Architectures de circuits (HSRA, RAW)
- Articulation processeur/reconfigurable (Garp...)
- Langages, programmation (Jhdl, Jbits ...)
- ...

Slide 9

1.7 Nos travaux: présentation générale

- Objectifs: programmer *vite*, concevoir une chaine *portable*, définir un *modèle de programmation* pour les architectures reconfigurables,
- Plan de travail: considérer l'architecture reconfigurable de manière abstraite, construire la chaine de développement en environnement objet pour permettre l'interaction verticale des outils.

Slide 10

2 Modélisation des architectures reconfigurables

Méthode objet : analyse de domaine puis abstraction et outils.

2.1 Phase analytique

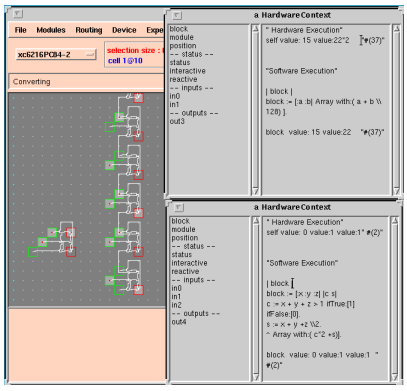
- Travaux de programmation/compilation sur FPGA (ArMen)
- Production d'un environnement de développement pour une architecture ouverte (xc6200), intégration du MAPP,
- Généralisation, abstraction.

Slide 11

2.2 Outils 6200

- Modélisation spécifique, outils de dessin structuré, place/route, support des blocs de haut niveau, support carte PCI. *Loic Lagadec*

Slide 12



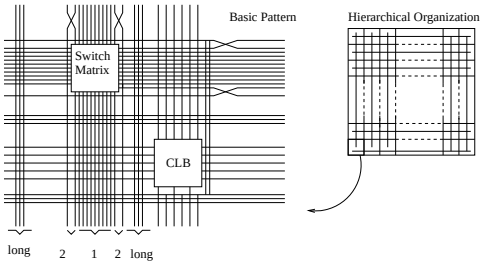
Slide 13

2.3 Modèle objet

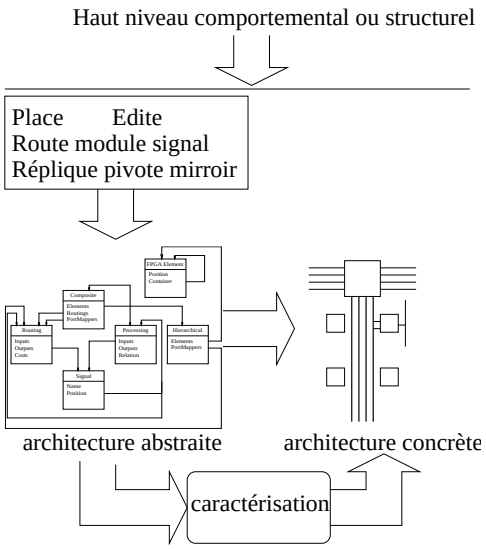
- Architecture reconfigurable abstraite : modèle hiérarchique objet appuyé sur une grammaire
- Meta-outils : manipulations de base sur l'architecture (placement, routages, dessin régulier),
- Description d'une architecture reconfigurable concrète : via des outils ou la grammaire. Possibilité de description paramêtrée et d'intégration de ressources spécifiques (mémoires, capteurs...).

Slide 14

Exemple d'architecture



Slide 15



Slide 16

3 Synthèse élémentaire

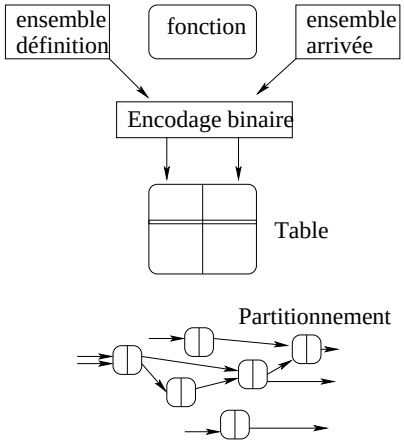
3.1 Modèle d'exécution

Le modèle d'exécution *élémentaire* est représenté à haut niveau par les hypothèses suivantes:

- Décomposition explicite des calculs sur des fonctions à grain fin synthétisables.
- Calcul de tables de look-up pour ces fonctions.
- Encodages des données (objets) et production de tables logiques.
- Partitionnement pour les LUT de la technologie.

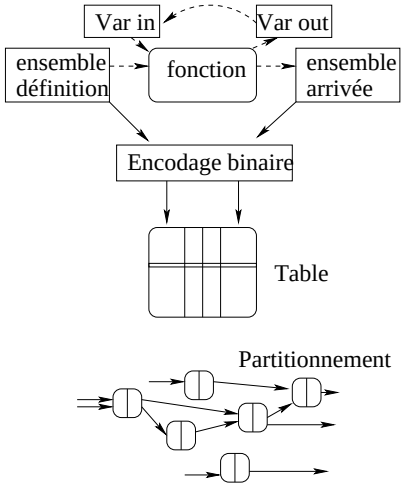
Slide 17

3.2 Synoptique combinatoire



Slide 18

3.3 Synoptique séquentiel



Slide 19

3.4 Considérations de portabilité

- Partitionnement : prendre en charge le nombre de variables accédant à chaque noeud,
- Placement : allouer les ressources logiques et les registres de l'architecture en fonction des caractéristiques des noeuds.

Slide 20

4 Modèles applicatifs

4.1 Outils de descriptions

- Formulation grammaticale : hiérarchies d'objets atomiques ou composites.
- Atomes : allocation directe dans l'architecture,
- Composites : réguliers (réseaux) ou irréguliers. Interconnexions d'atomes ou de composites.

Le modèle applicatif est simulable et synthétisable.

Slide 21

4.2 Descriptions structurelles

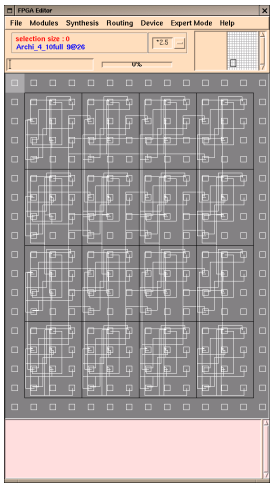
- Modélisation dans des classes,
- Description explicite d'éléments logiques maillés,

Ces descriptions sont synthétisables sur des architectures concrètes en s'appuyant sur les outils de partitionnement et de placement routage.

Exemples: construction d'opérateurs arithmétiques par des étudiants de Licence.

Slide 22

Multiplieur de Braun



Slide 23

5 Outils de bas niveau

Ensemble d'algorithmes génériques reconnus ou améliorés.

5.1 Partitionnement logique

- Utilisation de l'outillage implanté dans SIS (UCBerkeley),
- Echanges de descriptions basés sur les formats Espresso et BLIF
- Parallélisation de la synthèse (L.Lemarchand)

Slide 24

5.2 Partitionnement des graphes logiques

Laurent LeMarchand

- Génération d'un graphe de portes,
- Partitionnement de ce graphe en sous-graphes,
- Synthèse locale (plus efficace et parallélisable),
- Reconstitution d'une solution globale.

Slide 25

5.3 Principes des outils technologiques

Loic Lagadec

- Interfaces fonctionnels au niveau abstrait,
- Récupération des caractéristiques physiques dans les architectures concrètes,
  - Structures hiérarchiques de motifs,
  - Discontinuités,
  - Connectivités, ressources matérielles.
  - Fonctions de couts,

Slide 26

5.4 Outils

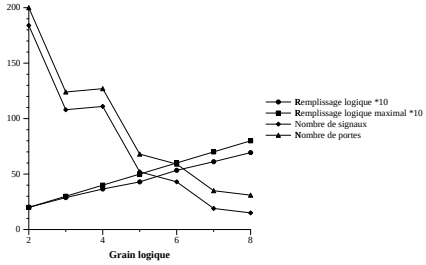
Notion de module physique associé à un graphe logique.

- Placeurs/routeurs : graphe de noeuds, modules, point à point,
- Réplications, miroirs, pivotements de modules,
- Edition interactive,
- Feedback vers les outils de niveau supérieur,

Slide 27

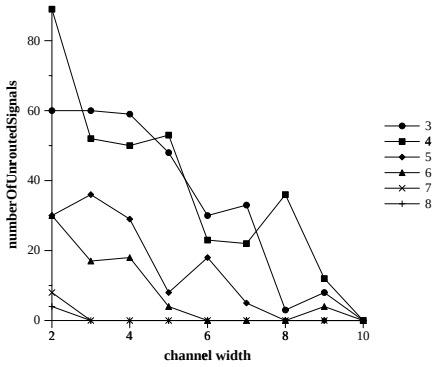
5.5 Investigations architecturales

Variation des grains logiques, des connectivités, tailles de canaux, chemins critiques.



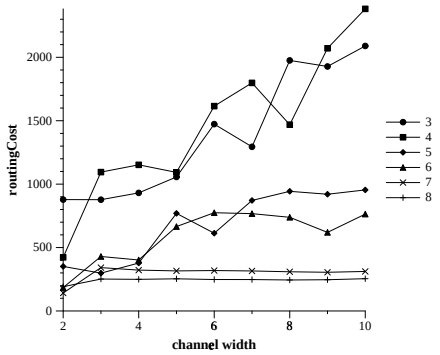
Slide 28

Difficultés de routage...



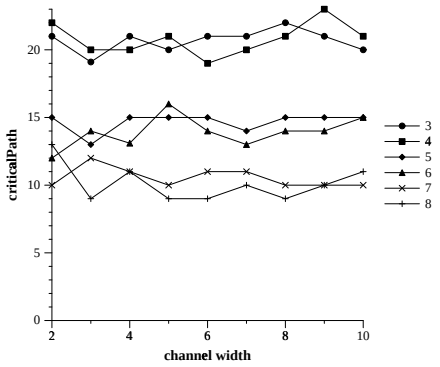
Slide 29

Couts du routage...



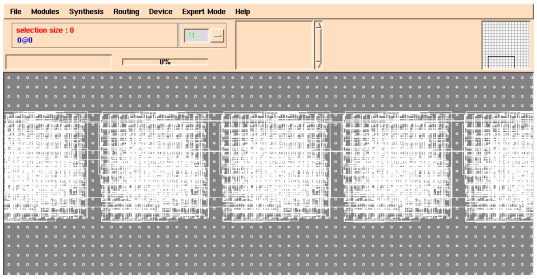
Slide 30

Chemins critiques ...



Slide 31

Exemple investigué: processeur DNA



Slide 32

6 Programmer pour le reconfigurable

Position actuelle:

- Floor-planner modulaire en cours de développement,
- Représentation des architectures limitée,
- Recherche sur la compilation vers la logique,

Slide 33

6.1 Types et synthèse

- Possibilité et nécessité d'implémentation *plus spécifique* qu'en VLSI,
- Vitesse et cout dépendent de l'optimisation logique,
- La qualité de la synthèse logique dépend de la précision obtenue sur les données.

Le mécanisme de synthèse basé sur les LUT se prète naturellement à la mise en place de l'inférence de types.

Slide 34

6.2 Types et compilation

- Langages typés: édition statique des liens, production de code basée sur des types simples spécifiés par le programmeur,
- Langages polymorphes: édition dynamique, le code est réutilisable indépendament des données,
- Autre solution: polymorphisme et propagation des spécifications de typage à la compilation.

La programmation objet autorise la disponibilité d'arithmétiques multiples, et l'adaptation de l'arithmétique aux problèmes.

Slide 35

Exemple d'un polynome

```
pol: x
^ ( x raisedTo: 3) + (3 * x )

testPol
| results anObject |
results := OrderedCollection new.
anObject := Something new.
results add: (anObject pol: 1).
results add: (anObject pol: 1.5).
results add: (anObject pol: 3 / 2).
results add: (anObject pol: 1.01s).

"OrderedCollection (4 7.875 (63/8) 4.06s)"
```

Slide 36

6.3 Définition des types

- Collection de valeurs (ensembles, intervalles, ...)
- Classes (Character, SmallInteger, Float134, GF16, DNA...)

6.4 Opérations sur les types

- unions, intersections, soustractions,
- arrondis, énumérations

Slide 37

7 Un compilateur objet et ses opérations

Analyse. production d'un graphe data-flow hiérarchique (HDFG)

Inference. propagation des types entrants dans le graphe

Calculs sur constantes. Intégration des constantes dans les messages.

Code mort. Suppression des sous-graphes non utilisés.

Factorisation. Détection des sous-expressions communes.

LUTification. Production de tables précalculées.

No-op . Suppression des messages sans effet.

Fusion. Sur observation de l'amplitude de données, regroupement de noeuds.

Slide 38

7.1 Étude de cas

Oscar Villellas

Encodeur/Décodeur Reed-Solomon pour un système RAID.

- Arithmétique sur un corps de Galois à 16 éléments
- Possibilité de gérer n+m disques ($n + m < 16$).
- Correction spécifique pour chaque disque en panne (décodeur).
- Encodage permanent vers les disques redondants.

Problème intéressant pour le reconfigurable.

Slide 39

7.2 Modélisation

- Processus d'encodage et de décodage RS basé sur un système linéaire. Smalltalk-80.
- Test et validation. Arithmétique standard Smalltalk-80.
- GF16. Discussion sur l'encodage des éléments. Smalltalk-80
- Test de RS sur GF16.

1 journée 1/2 de travail très conceptuel.

Slide 40

7.3 Compilation : résultats moyens sur 4:3

After compiler operation	operators	average input size	critical path
type inference	85.08	2	11.24
constant folding	11.68	2	7.65
dead-code removal	11.68	2	7.65
code factorization	10.41	2	7.65
operator to LUTs	10.41	1.43	7.65
no-op removal	7.42	1.65	5.75
operator fusion	4.5	2	3.625

Table 1: statistics for decoders-RS4:3.

Slide 41

7.4 Compilation : cas spécifique 8:2

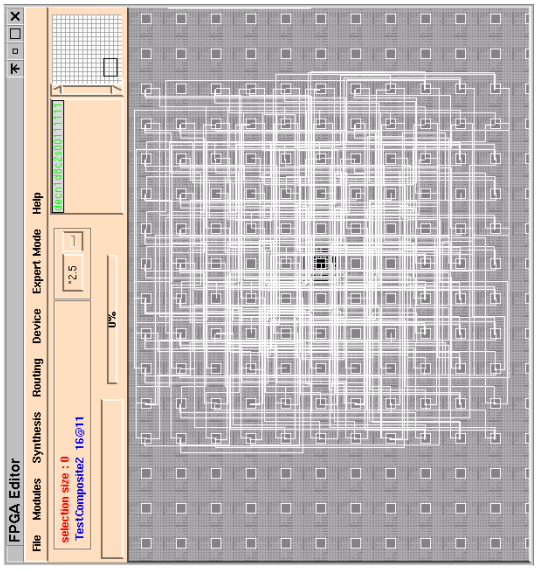
Compiler operation	operators	average input	critical path
type inference	90	2	15
constant folding	30	2	12
dead-code removal	30	2	12
code factorization	30	2	12
LUTification	30	1.47	12
no-op removal	21	1.67	9
operator fusion	14	2	8

Table 2: disk 1 decoder, with disks 1 and 2 broken

Slide 42

7.5 Synthèse

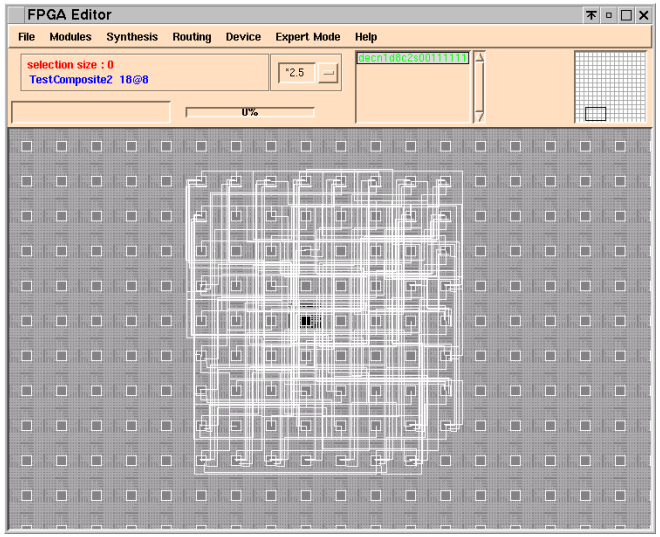
	L-2(enc)	L-4(enc)	L-2(dec)	L-4(dec)
Area	90	56	121	72
Input cells	32	32	40	40
Internal cells	53	**21**	79	**31**
Gates Input av.	2.0	3.62	2.0	3.81
Critical Path	18	14	19	15
CPU Time	43.14	20.34	98.70	34.89
Max struct. area	128	88	208	176
Cells used	109	85	181	170
Internal cells	53	29	79	58



Slide 43

8 Conclusion

- Aisance pour construire des outils de haut niveau, due au caractère ouvert des modélisation existantes
- Intéret des techniques de compilation (générales!) à confirmer,
- Possibilité de passer facilement de code microprocesseur à FPGA,
- Outil d'investigation pour les nouvelles architectures reconfigurables,
- Recherche algorithmique ouverte: on peut réutiliser quantité de code existant.


FPGA Editor
File
Modules
Synthesis
Routing
Device
Expert Mode
Help
selection size : 0
TestComposite2 18@8
*2.5
decn1d8c2s00111111
0%