Modélisation outillée d'architectures reconfigurables supportant les blocs IP matériels

Boutheina MAALOUL[1], Erwan FABIANI[1], Loïc LAGADEC[2]

[1]Univ. Bretagne Occidentale, UMR 6285, Lab-STICC, F-29200 Brest, France

[2] ENSTA Bretagne, UMR 6285, Lab-STICC, F-29200 Brest, France

boutheina.maaloul@univ-brest.fr, erwan.fabiani@univ-brest.fr, loic.lagadec@ensta-bretagne.fr

Résumé – Cet article présente une approche haut niveau et automatique d'intégration de blocs IP matériels adaptés aux défaillances dans le flot de conception d'un cadre logiciel de prototypage d'architecture reconfigurable. Ce travail est réalisé dans le cadre du projet ARDyT qui a pour objectif le développement d'un environnement complet pour la conception d'une architecture tolérante aux fautes dans le domaine aéronautique ou spatial avec un faible coût.

Abstract – This paper presents a high-level and automatic approach for the integration of hard IP blocks adapted to failures in a reconfigurable architecture prototyping software framework. This work is a part of the ARDyT project that aims to develop a complete environment for the design of a low cost fault tolerant architecture in aeronautical or space field. .

1 Contexte et problématique

Les architectures reconfigurables FPGA sont utilisées pour la mise en oeuvre de systèmes embarqués, en raison du compromis qu'elles offrent entre les hautes performances du matériel et la flexibilité du logiciel, dans de nombreux domaines d'application tels que l'automobile, l'aéronautique, le spatial et le nucléaire. Cependant, ces environnements sont soumis à des radiations (à des degrés de gravité divers) qui peuvent engendrer des fautes transitoires ou permanentes dans les circuits intégrés, impactant les composants de calcul, de mémorisation ou de configuration. Ces fautes peuvent entraîner une panne du système avec des conséquences catastrophiques en termes de sécurité et de pertes économiques.

Plusieurs stratégies existent pour remédier à ce problème. La première est le durcissement matériel des composants électronique pour les rendre moins sensibles aux radiations et robuste en environnement hostile. Cependant le processus de développement de ces circuits est long et coûteux ce qui limite leur utilisation. Une autre stratégie vise l'intégration de techniques de tolérance aux fautes dans les architectures reconfigurables, ce qui implique la mise en oeuvre de fonctionnalités telles que la détection d'erreurs en cours de fonctionnement, la localisation des fautes, la restauration d'état antérieur, la réparation des défauts ou la redondance de ressources [1, 2].

Le projet ANR ARDyT (Architecture Reconfigurable Dynamiquement Tolérante aux fautes) [3] propose le développement d'un environnement complet pour la conception d'une architecture tolérante aux fautes auto-adaptable aux environnements hostiles avec un coût faible par rapport aux architectures durcies. Cela nécessite le développement d'une architecture, de son environnement de programmation et des méthodologies de diagnostic et de testabilité en vue de l'amélioration de la fiabilité.

La conception et le développement d'un nouveau matériel passe par la réalisation de prototypes pour le test et la validation qui nécessite des coûts importants. La conception de prototypes virtuels basée sur des méta-modèles offre l'avantage de concentrer les efforts sur la modélisation et la conception des outils de CAO en réduisant le temps du cycle de développement.

Le cadre logiciel Drage est utilisé pour le prototypage virtuel d'architectures reconfigurables[4, 5]. Cependant une extension de ce cadre logiciel est obligatoire pour la mise en place de techniques de tolérance aux fautes. La première étape est la conception d'une méthode d'intégration d'opérateurs spécifiques (blocs IP matériels) dans le flot de l'outil.

2 Le cadre logiciel Drage

2.1 Fonctionnalités

Drage (Dynamic Reconfigurable Architecture Generation Environment) est un cadre logiciel développé en langage orienté objet non typé (Smalltalk) qui offre un environnement de prototypage complet d'architectures reconfigurables (voir figure 1) :

– La modélisation d'architectures reconfigurables homogènes ou hétérogènes à grains fin composées de LUT (Look Up Table) ou gros grains composées d'ALU ou bloc DSP à partir d'une description ADL (Architecture Description

Language) basée sur une grammaire spécifique.
- La génération automatique du code VHDL de l'architecture modélisée afin de permettre sa synthèse et son émulation sur FPGA (architecture virtuelle) ou, après ciblage de la technologie, de fondre le circuit ASIC correspondant.
- Le placement routage d'applications sur les architectures modélisées à partir d'une description hiérarchique RTL sous format BLIF, EDIF, ou un format interne de type CDFG et la génération du bitstream correspondant.
- La génération d'un contrôleur de configuration en code VHDL. Ce contrôleur permet la configuration partielle et multi-contextes des architectures reconfigurable. Il prend en charge l'ordonnancement des pages de configuration dans les zones reconfigurables et l'adressage du bitstream.

Le prototypage d'une architecture sous Drage nécessite la définition de deux couche différentes (voir figure 2) :
- La couche de calcul modélise les ressources de calcul et de mémorisation (LUT, bloc reconfigurables, ALUs) et leur connexion par des canaux de routage et des matrices d'interconnexion.
- La couche de configuration modélise les ressources dédiées à la configuration de la couche de calcul : contrôleur de configuration micro-programmable, mémoire de configuration, canaux d'acheminement des bits de configuration.

L'intégration de blocs IP matériels a des impacts sur les deux couches.

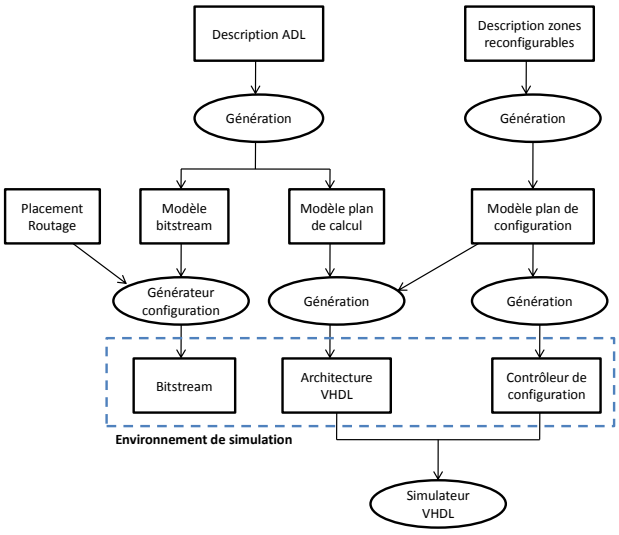

FIGURE 1 – Flot global de l'environnement logiciel Drage pour le prototypage d'architectures reconfigurables.

2.2 Génération automatique de VHDL

Une architecture reconfigurable est composée d'un pavage de tuiles composées d'éléments atomiques (luts, opérateurs, multiplexeurs, registres, etc.). La génération du code VHDL est réalisée après un parcours en profondeur des différents éléments. Conformément à la structure du language VHDL, l'outil génère le code correspondant à chaque élément (composé d'une entité et d'une architecture), le code qui décrit chaque tuile, les instructions d'appel aux composants atomiques et leur instanciation dans le code. La mémoire et le bus de configuration de l'architecture sont générés automatiquement en VHDL.

Après placement routage d'une application sur l'architecture décrite sous Drage, un bitsream est généré. Il est composé des mots de configuration de chaque tuile et généré après parcours en profondeur des différents éléments dans les zones de configuration définies par le concepteur. Ce bitstream est contrôlé par un contrôleur micro-programmable (machine à état finis) permettant l'ordonnancement des pages de configuration. L'acheminement du bitstream dans les zones reconfigurables est conditionné par les instructions du contrôleur définies manuellement par le concepteur de l'architecture. Les principales instructions sont la configuration, l'activation et la synchronisation des pages.

L'algorithme de génération des fichiers VHDL se basent sur le patron de conception objet "visiteur". Le fichier VHDL représentent la couche virtuelle qui peut être utilisée en entrée d'un simulateur (exemple Model Sim) ou synthétisée sur un FPGA commercial (voir figure 3). Dans ce dernier cas un processeur synthétisé pilote les instructions du contrôle de pages de configuration et l'interface des entrées/sorties de l'application (par exemple un Microblaze synthétisé sur FPGA)[4].

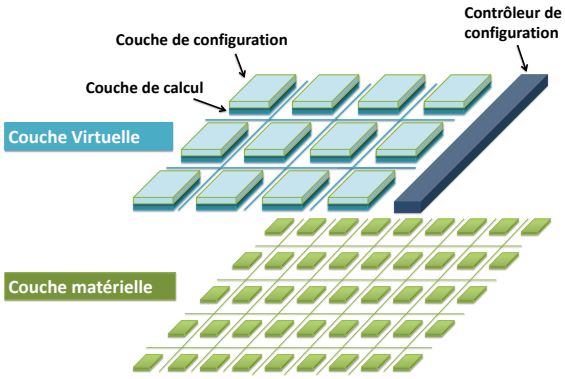

FIGURE 2 – Éléments constituant la couche virtuelle émulée sur une couche matérielle (FPGA).

3 Intégration de blocs IP matériels

3.1 Spécifications

Les blocs IP matériels à intégrer dans Drage sont des opérateurs logiques et/ou arithmétiques dotés d'une ou plusieurs fonctionnalités permettant la tolérance aux fautes. Dans le cadre du projet les fautes prises en compte sont les fautes transitoires résultante d'un SEU (Single Upset Event). Ces blocs ont une

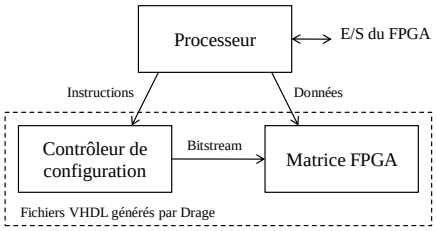

FIGURE 3 – Composants VHDL générés et interaction avec un processeur d'interfaçage.

ou plusieurs fonctionnalités parmi les suivantes :

- La détection des erreurs par la génération d'un ou plusieurs signaux permettant de vérifier la validité des résultats produits par exemple par le calcul du bit de parité ou le calcul du résidu. Une autre technique de détection est la réplication d'une fonction et la comparaison des résultats (Duplication With Detection). Une erreur est détectée si le comparateur détecte deux résultats différents.
- La correction des erreurs par l'utilisation de codes correcteurs d'erreurs (par exemple le code de Hamming).
- L'absorption des erreurs, via la redondance des éléments de calculs et l'utilisation de voteurs majoritaires, par exemple les méthodes de réplication de niveau 3 (Triple Modular Redundancy). Un voteur compare les résultats obtenus et le résultat majoritaire est considéré correct.

3.2 Modélisation

La généricité d'un point de vue développement est obligatoire dans notre cas afin d'intégrer un seul modèle qui permet d'instancier n'importe quel opérateur indépendamment de sa logique de fonctionnement.

La prise en compte des opérateurs intervient au niveau de la génération des couches de calcul et de configuration, et dans les outils de synthèse d'une configuration pour une application.

Un opérateur est défini par les informations suivantes (voir figure 4) :

- Son nom, tel que défini dans la description HDL.
- Ses entrées/sorties (noms et largeurs).
- L'énumération de ses fonctions implémentées et de leur code de configuration associé.
- Son comportement décrit dans un programme HDL.

Concrètement ces informations sont renseignées dans une description XML spécifique (voir Listing 1), et le comportement de l'opérateur est décrit en VHDL.

3.3 Intégration

L'intégration de la description d'un opérateur intervient à trois niveaux :

- Au niveau de la modélisation de la couche de calcul, l'intégration d'un opérateur dans un bloc de calcul est opérée par la spécification d'une interface d'entrées/sorties compatible, et la référence au fichier de description XML (voir Listing 2). Une instance de l'opérateur est créée automatiquement à partir de cette description. La génération du code VHDL de toute l'architecture établit le lien avec le programme VHDL de l'opérateur.
- Au niveau de la modélisation de la couche de configuration, la mémoire de configuration et le format de bitstream sont automatiquement adaptés au codage des fonctions de l'opérateur.
- Au niveau des outils d'allocation de ressources (pour la synthèse et le placement-routage d'une application), les noms des fonctions de l'opérateur sont automatiquement adjointes à l'ensemble des fonctions recherchées dans la description d'une application.

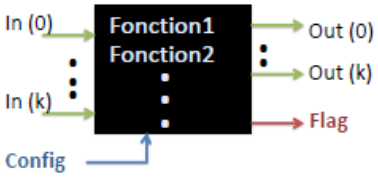

FIGURE 4 – Un bloc IP générique.

Listing 1 – Représentation XML d'un opérateur doté de fonctions arithmétiques et logiques.

```
< ?xml version="1.0" encoding="ISO-8859-1" ?>
< !DOCTYPE operator SYSTEM "Operator.dtd">
<operator name="ALU" acces="/home/Drage/alu.vhdl">
 <inputs>
  <port name="in1" width="8"/>
  <port name="in2" width="8"/>
  <portConfig name="selFunc" portWidth="3" nbrFunc= "8">
    <function name="add" configBits="000"
    protection="parity"/>
    <function name="sub" configBits="001"/>
    <function name="mul" configBits="010"/>
    <function name="sll" configBits="011"/>
    <function name="not" configBits="100"/>
    <function name="and" configBits="101"/>
    <function name="or" configBits="110"/>
    <function name="xor" configBits="111"/>
   </portConfig>
 </inputs>
 <outputs>
  <port name="out" width="8"/>
 </outputs>
</operator>
```

Listing 2 – Syntaxe de modélisation d'un opérateur sous Drage

```
((ARRAY
  (DOMAIN 1 1 50 50) "END of DOMAIN"
   ((COMPOSITE
( ( ((BLOCKCAL
  (INPUTS
   (( WIRE ( WIDTH 8) ) NAMED in1 )
   (( WIRE ( WIDTH 8) ) NAMED in2 ) )
  (OUTPUTS (( WIRE ( WIDTH 8) ) NAMED out ) )
  (FILE '/home/test/operator.xml' ) )
...
```

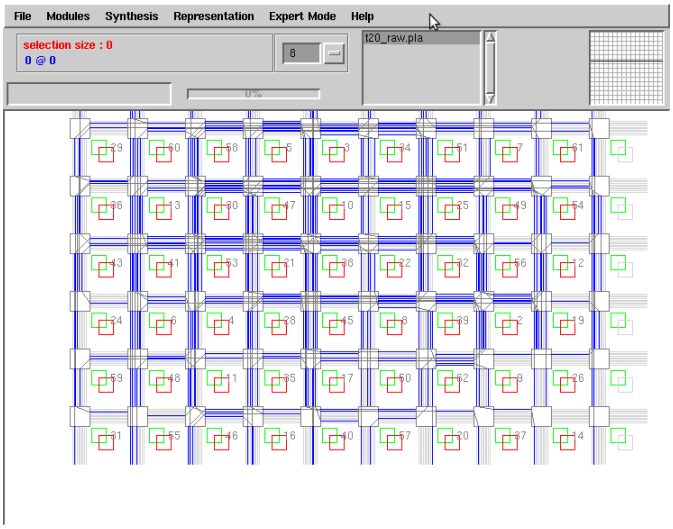

FIGURE 5 – Visualisation des canaux de routage d'une application sur une architecture qui intègre des ALUs tolérantes aux fautes.

4 Résultats

La méthode d'intégration a été validée via la modélisation sous Drage d'une architecture composée d'un pavage régulier d'ALUs adaptées aux erreurs (bit de parité) et de LUTs. Nous avons placé-routé sur cette architecture une application de test composée d'un graphe d'opérations arithmétiques. La figure 5 montre l'IHM de visualisation de la structure de l'architecture et des ressources utilisées (blocs de calcul et canaux de routage). Drage permet aussi de générer le VHDL de l'architecture reconfigurable modélisée. Par exemple, pour une matrice de 10x10 tuiles, définie par 42 lignes de code de la grammaire Drage, 10188 lignes de code VHDL sont générées.

L'automatisation de l'intégration des opérateurs permet ainsi de réduire le temps et les efforts de développement. Elle ouvre la voie à la prospection architecturale pour la comparaison des opérateurs, pour évaluer la meilleure stratégie d'adaptation aux erreurs, notamment en terme de ressources utilisées et d'adéquation avec des algorithmes du domaine spatial.

5 Conclusion et perspectives

Dans cet article nous avons présenté une approche d'intégration de blocs IP matériels tolérants aux fautes. Cette intégration doit se poursuivre par la mise en oeuvre d'une méthode de désignation des configurations pour limiter les erreurs dûes à des perturbations de la mémoire de configuration, en augmentant la distance entre deux mots de configuration, et en modifiant dynamiquement le code VHDL de l'opérateur.

Après la phase d'intégration des opérateurs, la modélisation d'une couche d'introspection permettra la mise en oeuvre de la surveillance et le traçage des signaux interne de l'architecture (notamment les signaux d'alerte provenant des opérateurs), sans utiliser de ressources de la couche de calcul. Elle permettra aussi la relecture continue du bitstream pour détecter la présence d'une erreur dans les mémoires de configuration et de recharger le bitstream correct dans la zone concernée.

Remerciements

Ce travail a été soutenu par le projet ANR ANR-11-INSE-15 "Architecture Reconfigurable Dynamiquement Tolérante aux fautes (ARDyT)".

Références

[1] F.L. Kastensmidt, L. Carro et R. Reis. *Fault-Tolerance Techniques for SRAM-based FPGAs*, Springer, 2006.

[2] N. Battezzati, L. Sterpone et M.Violante. *Reconfigurable Field Programmable Gate Array for Mission-Critical Application*, Springer, 2011.

[3] ArdyT consortium. *http ://www.ardyt.irisa.fr*.

[4] D. Picard et L. Lagadec. *Fast prototyping environment for embadded reconfigurable units*, Reconfigurable Communication-centric System-on-Chip (ReCoSoc), 6th International Workshop, 2011.

[5] L. Lagadec et D. Picard. *Teaching reconfigurable processor : the biniou approach*, KIT Scientific report, 7551, isbn : 978-3-86644-515-4, 2010.