HAL
archives-ouvertes.fr

Plateforme matérielle–logicielle à bas coût pour l'émulation de fautes

Pierre Guilloux, Arnaud Tisserand

▶ **To cite this version:**

Pierre Guilloux, Arnaud Tisserand. Plateforme matérielle–logicielle à bas coût pour l'émulation de fautes. Colloque du GDR SoC-SiP, Jun 2016, Nantes, France. hal-01346576

HAL Id: hal-01346576
https://hal.inria.fr/hal-01346576

Submitted on 19 Jul 2016

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.

Copyright

Plateforme matérielle–logicielle à bas coût pour l'émulation de fautes

Pierre Guilloux [3,1] et Arnaud Tisserand [2,1]
[1]IRISA, [2]CNRS, [3]Université Rennes 1, INRIA, 6 rue de Kerampont, 22305 Lannion.

Abstract

Nous présentons les premiers développements d'une plateforme matérielle–logicielle d'émulation de fautes basée sur un réseau de cartes FPGA avec processeurs multicœurs embarqués et un serveur pour les outils de CAO.

1 Introduction

Évaluer finement des architectures de détection ou de tolérance aux fautes est une tâche souvent complexe en particulier pour les circuits arithmétiques où l'impact des fautes sur l'erreur mathématique nécessite de très nombreuses simulations [2]. On recourt souvent à des simulations de bas niveau, de type « bit précis et cycle précis », où on injecte successivement de nombreuses fautes dans le temps selon différents scénarios représentatifs. Cette technique purement logicielle est simple mais limitée par la vitesse des simulateurs. L'*émulation matérielle* sur FPGA est souvent utilisée pour accélérer de telles simulations.

2 Description de la plateforme

Comme dans la littérature, nous émulons des fautes de collage et d'inversion en insérant des injecteurs de fautes à des endroits choisis de la *netlist* comme illustré en figure 1.

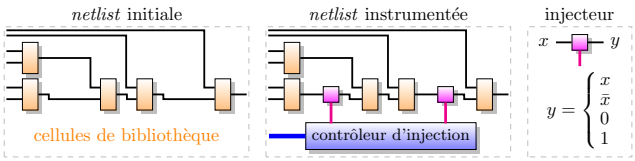

FIGURE **1. Principe des injecteurs de fautes.**

Notre plateforme, décrite en figure 2, comporte : 1 serveur Debian (8 cœurs 4.0 GHz, 32 Go RAM, 128 Go SSD, 1 To HDD, 2 GbE, coût 1100 €) ; des cartes FPGA Zynq Xilinx ; 1 *switch* ethernet 1 Gb/s (180 €) ; 1 carte d'alimentation électrique et de pilotage maison (200 €). Actuellement, les cartes FPGA sont : 3 Zedboard (Zynq 7020 à 250 € pièce) ; 1 PicoZed (Zynq 7035 à 810 €) ;

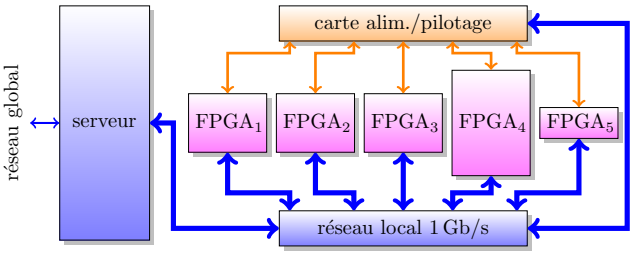

FIGURE **2. Architecture de la plateforme.**

1 Zybo (Zynq 7010 à 150 €). Le coût total de la plateforme, avec câblage et accessoires, est d'environ 3500 €HT.

La partie reconfigurable (PL, *programmable logic*) du Zynq va porter les *blocs d'émulation* contenant l'opérateur matériel évalué et l'instrumentation nécessaire à l'émulation de fautes. La partie processeur (PS, *processing system*) du Zynq va recevoir l'interface entre la PL et le serveur ainsi que la gestion à haut niveau de l'émulation (p. ex. gestion des fichiers de données/résultats). Un bloc d'émulation, décrit en figure 3, se compose des ressources matérielles :

- l'opérateur original, sans instrumentation ;
- l'opérateur instrumenté avec des injecteurs de fautes ;
- le contrôleur d'injection ;
- la mémoire et le générateur de vecteurs d'entrées ;
- l'unité d'analyse en-ligne ;
- le contrôle global du bloc d'émulation ;
- enfin, l'interface vers la PS.

L'analyse en-ligne peut être la simple détermination du taux de couverture de fautes (nb fautes détectées vs. nb fautes injectées), ou celle de la moyenne ou de la distribu-

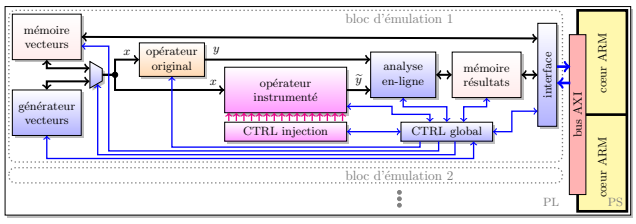

FIGURE **3. Architecture d'un bloc d'émulation.**

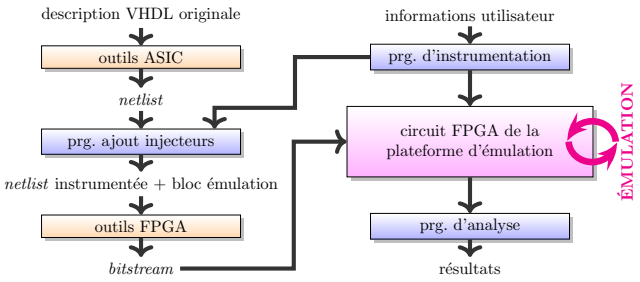

FIGURE **4. Flot d'outillage de la plateforme.**

tion de l'erreur mathématique de la sortie « fautée ».

Le flot logiciel, présenté en figure 4, s'exécute essentiellement sur le serveur. La description VHDL originale de l'opérateur est transformée par les outils ASIC commerciaux (Synposys en synthèse physique et Cadence en *back-end*), en une *netlist* de bas niveau. Cette *netlist* contient l'ensemble des cellules de bibliothèque utilisées à plat, c.-à-d. sans hiérarchie interne, et leurs connexions.

Notre programme ajoute des injecteurs de fautes à différents endroits de cette *netlist* pour produire la *netlist* instrumentée (répartition aléatoire sur les signaux internes ou une répartition aléatoire uniforme sur la surface). Il ajoute les autres éléments du bloc d'émulation de la figure 3. Le contenu du bloc d'émulation, en particulier le contrôleur d'injection de fautes, est généré en fonction des choix utilisateur sur les scénarios de fautes à considérer et les types de vecteurs d'entrées. Les vecteurs d'entrée proviennent de compteurs ou LFSR (*linear feedback shift register*) ou bien d'une mémoire stockant des données représentatives.

Plusieurs blocs d'émulation peuvent fonctionner en parallèle dans un même FPGA. Différentes solutions sont possibles comme utiliser un même fichier de données d'entrée diffusé à tous les blocs d'émulation et des scénarios différents ; ou bien utiliser des générateurs de vecteurs d'entrée différents et un même scénario de fautes.

Les blocs d'émulation sont synthétisés, placés/routés pour former la configuration du FPGA. Le *bitstream* est alors chargé sur le FPGA à travers le réseau et l'émulation est lancée. Enfin, les résultats de l'émulation sont envoyés au serveur pour l'analyse finale. La partie logicielle sur la PS du circuit Zynq gère la récupération du *bitstream*, le contrôle de haut niveau de l'émulation dans chaque FPGA et les échanges de données avec le serveur.

Le fait d'avoir plusieurs cartes FPGA permet de recouvrir les temps d'émulations et ceux des transferts de données sur le réseau. Nous déployons sur la plateforme un système libre de gestion et d'ordonnancement de tâches parallèles pour grappe de machines en utilisant SLURM (*Simple Linux Utility for Resource Management* `http://slurm.schedmd.com/`), mais il nous reste encore beaucoup de travail sur cet aspect.

3 Exemple d'application

Notre exemple est un simple multiplieur entier 16 bits non signé (issu de [3]) avec code résidu [2, 1] modulo $m \in \{3, 7, 15\}$ utilisant la relation :

$$(a \times b) \bmod m \;=\; \big((a \bmod m) \times (b \bmod m)\big) \bmod m$$

Par rapport à une simulation RTL (Vivado 2014.4) sur le serveur, un seul bloc d'émulation sur un seul FPGA permet un facteur d'accélération de plus de $\times 6880$.

La figure 5 présente la distribution d'erreur obtenue pour différentes positions aléatoires du signal interne fauté (une par couleur sur la figure) avec un test exhaustif des 2^{32} entrées possibles en seulement quelques minutes.

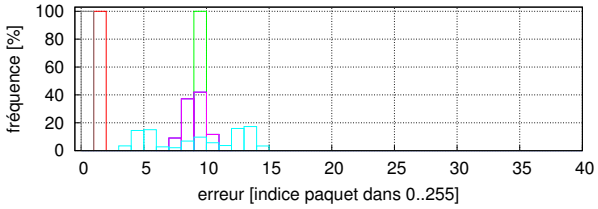

FIGURE **5. Distribution de l'erreur mathématique pour différents scénarios de fautes (attention seuls les 40 premiers paquets d'erreur sont représentés en abscisse sur les 256 mesurés, au-dessus ils sont tous à 0).**

Ce travail est encore en développement, il nous reste beaucoup à faire sur la plateforme et son exploitation.

Remerciements

Ce travail a été financé en partie par les projets ARDyT (ANR-11-INSE-15, `http://ardyt.irisa.fr/`) et Reliasic (Labex CominLabs, `http://www.reliasic.cominlabs.ueb.eu/`). Nous remercions l'aide du *Xilinx University Program*.

Références

[1] K. Bigou, T. Chabrier, and A. Tisserand. Opérateur matériel de tests de divisibilité par des petites constantes sur de très grands entiers. In *Actes Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS)*, Grenoble, France, Jan. 2013.

[2] I. Koren and C. M. Krishna. *Fault-Tolerant Systems*. Morgan-Kaufman, San Francisco, CA, USA, 2007.

[3] R. Zimmermann. VHDL library of arithmetic units. In *Proc. 1st Forum on Design Languages (FDL)*, pages 267–272, Lausanne, Switzerland, Sept. 1998. `http://www.iis.ee.ethz.ch/~zimmi/arith_lib`.