HAL open science

Protection sélective d'un réseau de neurones implémenté sur puce FPGA soumis à un environnement radiatif

Wilfread Guilleme, Youri Helen, Rémy Priem, Angeliki Kritikakou, Daniel Chillet, Cédric Killian

To cite this version:

Wilfread Guilleme, Youri Helen, Rémy Priem, Angeliki Kritikakou, Daniel Chillet, et al.. Protection sélective d'un réseau de neurones implémenté sur puce FPGA soumis à un environnement radiatif. Gretsi 2023 - XXIXème Colloque Francophone de Traitement du Signal et des Images, Université Grenoble Alpes, Aug 2023, Grenoble, France. pp.1-4. hal-04396267

HAL Id: hal-04396267

https://inria.hal.science/hal-04396267v1

Submitted on 16 Jan 2024

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.

Distributed under a Creative Commons Attribution 4.0 International License

Protection sélective d'un réseau de neurones implémenté sur puce FPGA soumis à un environnement radiatif

Wilfread GUILLEMÉ[1] Youri HELEN[2] Rémy PRIEM[2] Angeliki KRITIKAKOU[1] Daniel CHILLET[1] Cédric KILLIAN[1]

[1]Université de Rennes, Inria, IRISA, 263 Av. Général Leclerc, 35000 Rennes, France

[2]DGA MI, 136 La Roche Marguerite, 35170 Bruz, France

Résumé – Les applications avioniques sont confrontées à de nombreux défis pour garantir un fonctionnement sûr et fiable. Elles doivent être capables de fonctionner avec des environnements contraints dans lesquels les circuits sont soumis à l'effet des particules ionisantes ou des rayons cosmiques. En outre, ces systèmes requièrent des traitements de plus en plus complexes, ce qui a conduit à l'exploration de solutions telles que l'utilisation de réseaux de neurones embarqués sur FPGA. Afin de répondre à la problématique d'exécution de ces algorithmes vis-à-vis des fautes environnementales, une méthode de protection est proposée. Elle est basée sur la triplication des bascules détectées comme sensibles à l'issue d'une simulation. Les résultats de l'étude indiquent que la phase d'apprentissage impacte la sensibilité des différentes couches du réseau de neurones. Les bits de signe et de poids fort qui stockent le résultat des calculs des neurones artificiels sont particulièrement vulnérables et doivent être protégés. La stratégie de protection sélective présentée dans cet article permet d'augmenter la tolérance aux fautes de ces systèmes en offrant un compromis concernant la consommation de ressources matérielles.

Abstract – Avionics applications face many challenges in ensuring safe and reliable operation. They must be capable of functioning in constrained environments where circuits are subjected to the effects of ionizing particles or cosmic rays. In addition, these systems require increasingly complex processing, which has led to the exploration of solutions such as the use of neural networks embedded on FPGAs. To address the issue of executing these algorithms in the face of environmental faults, a protection method is proposed. It is based on triplicating flip-flops detected as sensitive after a simulation. The study's results indicate that the learning phase impacts the sensitivity of different layers of the neural network. The sign and strong weight bits that store the results of artificial neuron calculations are particularly vulnerable and must be protected. The selective protection strategy presented in this article increases the fault tolerance of these systems by offering a compromise regarding hardware resource consumption.

1 Introduction

Les systèmes embarqués dans les applications avioniques doivent faire face à de nombreux enjeux pour garantir un fonctionnement sûr et fiable. Ils doivent être en mesure de supporter des environnements difficiles. Parmi ces perturbations, les effets des radiations ionisantes peuvent causer des erreurs critiques dans les circuits électroniques, compromettant ainsi le fonctionnement du système. Dans ce contexte, la protection des réseaux de neurones embarqués devient un enjeu crucial. Ces algorithmes peuvent être utilisés dans les applications avioniques pour la reconnaissance d'objets ou le contrôle de vol. L'impact d'une particule ionisante sur le circuit peut provoquer un Single Event Upset (SEU) [1] qui se caractérise par l'inversion d'un bit contenu dans une bascule. Dans certains cas, une telle erreur peut entraîner une mauvaise décision du système avionique en raison d'une erreur de prédiction du réseau de neurones.

L'implémentation d'un réseau de neurones sur FPGA présente de nombreux avantages. En effet, ces puces permettent de réaliser des calculs en parallèle, ce qui accélère considérablement le temps de calcul. De plus, ils sont conçus pour être économes en énergie, ce qui permet de réaliser des calculs complexes avec une bonne efficacité énergétique. Parmi les différentes familles de FPGAs, la technologie Flash est la plus adaptée aux applications avioniques. En effet, dans ces FPGAs, la configuration est stockée dans des mémoires non volatiles, moins sensibles aux effets des radiations que pour la technologie SRAM. Pour compléter le niveau de durcissement d'un système électronique, il est possible de le protéger vis-à-vis des fautes via différentes techniques, comme la triplication (TMR) [2], ou la redondance d'informations par les codes correcteurs d'erreurs Hamming [3] et BCH [4]. Dans le cas des réseaux de neurones, il est important de noter que ces algorithmes sont intrinsèquement résilients aux fautes et aux approximations de calcul [5]. Il peut donc être judicieux de ne protéger que les éléments sensibles du système dans l'objectif d'économiser de la ressource matérielle.

Cet article a pour objectif de présenter une méthode de protection d'un perceptron multicouche [6] permettant la reconnaissance des chiffres. Ce réseau de neurones a été développé en Python puis implémenté en VHDL pour ensuite être programmé sur une carte de développement. Dans un premier temps, l'architecture du réseau de neurones est décrite. Le scénario de test permettant de détecter les bascules sensibles est ensuite expliqué. Les résultats obtenus sont justifiés par des études portant sur la résilience de l'algorithme en fonction du nombre d'itérations de la phase d'apprentissage, la vulnérabilité des bascules des neurones et la sensibilité des différentes couches. Enfin, une évaluation des ressources requises pour protéger l'architecture est décrite.

2 Architecture du réseau de neurones

Cette section présente l'architecture du perceptron multicouche étudié. Elle décrit le contexte de la phase d'apprentissage ainsi que les différences qui existent entre la conception du modèle et son implémentation matérielle.

2.1 Phase d'apprentissage

Le réseau de neurones étudié est un perceptron multicouche dont l'objectif est de reconnaître des chiffres. Il a été développé à l'aide de Keras/TensorFlow [7] en utilisant la base de données MNIST [8] pour réaliser l'apprentissage. Ce dataset est constitué de 70 000 images de chiffres manuscrits, dont 60 000 sont destinées à l'entraînement du modèle et 10 000 pour le test. Ces images, à l'origine de taille 28x28 en nuances de gris ont été modifiées en images 8x8 en noir et blanc, elles constituent les 64 entrées du réseau de neurones. Cette phase d'apprentissage permet d'estimer les poids et les biais du modèle.

2.2 Architecture du modèle

La figure 1 présente l'architecture du perceptron multicouche étudié, elle montre que la couche d'entrée du perceptron est de taille 64, qu'il est constitué d'une couche intermédiaire avec 64 neurones ainsi qu'une couche de sortie avec 10 neurones pour représenter les chiffres de 0 à 9.

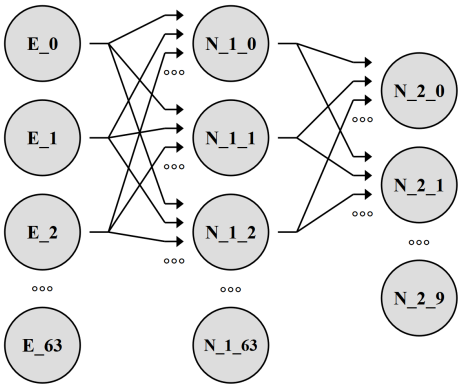

FIGURE 1 : Architecture du perceptron multicouche.

La fonctionnalité d'un neurone artificiel, comme le montre la figure 2 repose sur un bloc de logique combinatoire qui effectue l'opération multiplication accumulation entre les entrées Xi et les poids Wi, un biais B est ajouté. Le résultat est ensuite stocké dans un registre de bascules avant d'être envoyé vers la mémoire qui modélise la fonction d'activation sigmoïde.

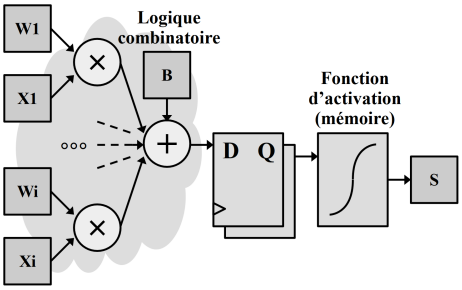

FIGURE 2 : Schéma fonctionnel d'un neurone artificiel.

2.3 Représentation des données sur FPGA

Les données et les paramètres d'un réseau de neurones sous l'environnement Python sont des nombres réels et signés. Ces valeurs sont instanciées dans le format à virgule flottante simple précision codé sur 32 bits. Cet encodage n'est pas optimal pour les FPGA. En effet, la ressource matérielle y est beaucoup plus limitée. Dans l'environnement FPGA, les données sont représentées par le type SFIXED [9] qui permet de représenter un nombre réel signé et d'obtenir un format de la donnée sur-mesure. Le bit de poids fort correspond au signe, les bits suivants correspondent à la partie entière puis à la partie décimale avec une taille indiquée. La donnée est représentée en utilisant la méthode de complément à deux. L'encodage SFIXED choisi illustré à la figure 3 est constitué de 10 bits dont 1 pour le signe, 4 pour la partie entière et 5 pour la partie décimale.

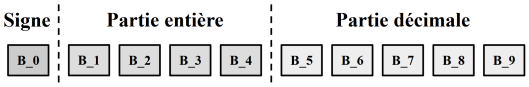

FIGURE 3 : Format de données SFIXED étudié.

3 Détection des bascules sensibles

Cette section présente la solution matérielle permettant l'injection d'une faute sur un neurone, l'architecture du système permettant la détection des bascules sensibles d'un perceptron multicouche ainsi que les données d'entrée utilisées.

3.1 Injection d'une faute sur un neurone

Afin de pouvoir détecter les bascules sensibles de l'architecture, il est nécessaire de développer une solution permettant d'émuler un SEU au niveau des neurones. Ils doivent être modifiés comme indiqué en figure 4 en ajoutant des commandes supplémentaires permettant l'inversion du contenu d'une bascule. La sélection de la faute à injecter est assurée par un multiplexeur et des portes inverseuses.

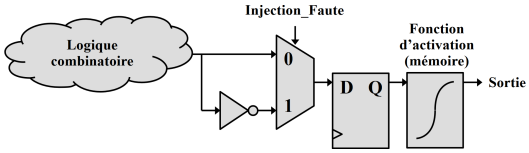

FIGURE 4 : Injection d'une faute sur une bascule d'un neurone.

3.2 Architecture matérielle de la détection

L'implémentation matérielle de l'architecture de détection de faute est présentée en figure 5. Elle est constituée de deux réseaux de neurones identiques exécutés en parallèle, ils sont notés ANN1 et ANN2. Le premier modèle fonctionne normalement tandis que le second modèle est soumis au dispositif d'injection de fautes présenté précédemment capable de simuler un SEU sur une seule bascule sélectionnée.

Le résultat de prédiction des deux perceptrons multicouches est ensuite comparé. Si les prédictions fournies par les modèles pour toutes les combinaisons d'entrées possibles sont identiques, cela signifie que la bascule sélectionnée n'a pas

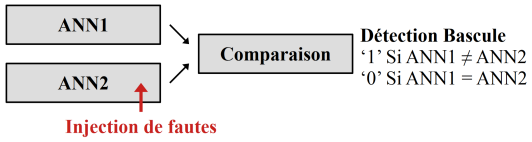

FIGURE 5 : Implémentation matérielle de la détection des bascules sensibles.

provoqué d'erreur de prédiction du modèle. Elle n'est donc pas considérée comme sensible.

À l'inverse, si les sorties des deux réseaux de neurones sont différentes pour des données d'entrée identiques, cela signifie que la bascule désignée a provoqué une erreur de prédiction du modèle et est donc considérée comme sensible.

3.3 Scénario de test

La section précédente a permis d'expliquer la procédure de détection d'une seule bascule sensible. La procédure, traduite par l'algorithme 1 permet d'inverser successivement le contenu des 10 bascules de chacun des 74 neurones pour 20 images différentes. L'exécution de cet algorithme permet de dresser une table de sensibilité des bascules.

Algorithme 1 : Détection des bascules sensibles pour un réseau de neurones sur FPGA

```
1  i = 0 (Selection de l'image)
2  n = 0 (Selection du neurone)
3  b = 0 (Selection de la bascule)
4  pour i = 0...19 faire
5      pour n = 0...73 faire
6          pour b = 0...9 faire
7              ANN1 = CalculANN(i);
8              InversionBascule(n, b);
9              ANN2 = CalculANN(i);
10             InversionBascule(n, b);
11             si ANN1 ≠ ANN2 alors
12                 BasculeSensible(n, b) + +
13             fin
14         fin
15     fin
16 fin
```

Cette méthode permet de définir un niveau de sensibilité pour chacun des éléments séquentiels. Par exemple, une bascule détectée comme sensible sur toutes les images du scénario de test obtiendra un niveau de sensibilité de 20. Une bascule détectée sur la moitié des images aura un niveau de sensibilité de 10. Ces différentes valeurs permettront de définir les bascules prioritairement à protéger en privilégiant celles ayant le plus grand niveau de sensibilité.

4 Résultats obtenus

Cette section présente les résultats obtenus au terme de différents scénarios de détection des bascules sensibles. Les études portent sur la vulnérabilité des bascules des neurones et la sensibilité des différentes couches. La résilience de l'algorithme en fonction du nombre d'itérations de la phase d'apprentissage. Enfin, une évaluation de la consommation de ressources en bascules de la méthode de protection proposée.

4.1 Sensibilité des couches et de la donnée

La figure 6 représente un tableau obtenu à l'issue du scénario de test de détection des bascules sensibles présenté à la section précédente. Il s'agit d'un modèle ayant effectué 1000 itérations d'apprentissage avec un batch_size de 32, qui est un hyperparamètre qui définit la taille de l'échantillon d'entrée qui est propagé à travers le réseau de neurones à chaque étape de la descente de gradient.

	Signe	Partie entière				Partie décimale				
Neurone	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9
N_1_0	2	1	2	0	1	1	0	0	0	0
N_1_1	4	0	1	0	0	0	0	0	0	0
N_1_2	5	1	0	0	0	0	0	0	0	0
N_1_3	2	1	1	1	0	0	0	0	0	0
N_1_4	3	1	2	1	1	0	0	0	0	0
...										
N_2_0	11	0	1	0	0	0	0	0	0	0
N_2_1	8	2	2	0	2	0	0	0	0	0
N_2_2	20	1	2	0	0	0	0	0	0	0
N_2_3	18	2	3	1	1	0	0	0	0	0
N_2_4	12	2	1	0	0	0	0	0	0	0
N_2_5	19	2	1	1	1	1	0	0	0	0
N_2_6	10	1	1	1	0	0	0	0	0	0
N_2_7	16	1	0	2	2	0	0	0	0	0
N_2_8	17	3	2	2	2	1	0	0	0	0
N_2_9	20	0	2	0	0	0	0	0	0	0

FIGURE 6 : Niveau de sensibilité des différentes bascules.

Il montre que les bascules contenant les bits de poids forts sont les plus sensibles. Celles qui contiennent les bits de signe le sont encore plus. Ces résultats sont expliqués par le fait qu'une inversion sur le signe ou un bit de valeur important change radicalement la donnée d'entrée pour la fonction d'activation ayant un impact fort sur le résultat de sortie du neurone.

Cette étude montre que la couche de sortie est beaucoup plus sensible que la couche intermédiaire. Ceci peut être expliqué par le fait qu'elle est constitué de moins de neurones et qu'il s'agit de la couche qui va définir le résultat de prédiction du modèle.

4.2 Résilience en fonction de l'apprentissage

Au cours de ce travail, plusieurs réseaux de neurones ont été générés. Il a été constaté que leur tolérance aux fautes variait en raison de la nature aléatoire de la phase d'apprentissage. Pour cette raison, une étude de la tolérance aux fautes en fonction du nombre d'itérations d'apprentissage a été menée. Les résultats de cette étude, présentés dans la figure 7, mettent en évidence deux évolutions distinctes du nombre de fautes pouvant provoquer une prédiction incorrecte du réseau de neurones.

La courbe bleue représente les résultats de la détection de fautes lors de l'utilisation des images d'entraînement, tandis que la courbe rouge représente les images de validation. Elles sont calculées en additionnant les niveaux de sensibilités des différentes bascules. Les paramètres d'un modèle ont été extraits à différents moments de l'apprentissage, qui sont représentés sur l'axe des abscisses. Les résultats indiquent que la tolérance aux fautes du réseau de neurones augmente avec le nombre d'itérations d'apprentissage. En outre, le modèle

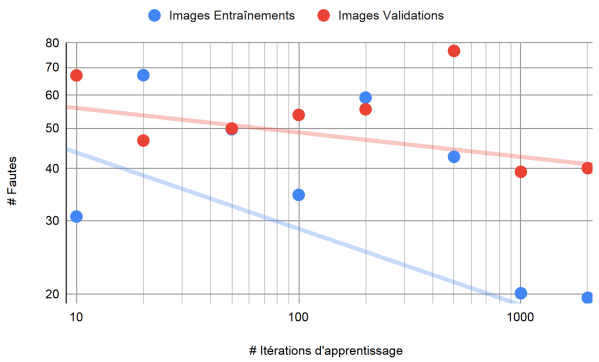

FIGURE 7 : Nombre de fautes détectées par le scénario de test.

est plus résilient lorsqu'il est exposé à un environnement pour lequel il a été conçu. Les images d'entraînements provoquent moins d'erreurs que les images de validations. Pour ce résultat, le modèle de réseau de neurones le plus résilient est celui ayant effectué 1000 itérations d'apprentissage.

4.3 Consommation de ressource matériel

La figure 8 illustre le nombre de bascules requis pour implémenter le réseau de neurones étudié, selon les stratégies de protection et les itérations d'apprentissage.

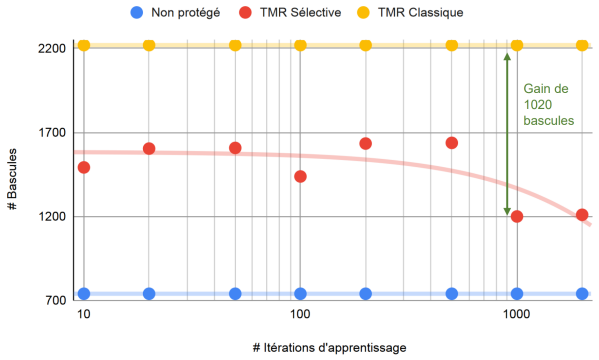

FIGURE 8 : Coût en bascules de la triplication sélective.

Avec 74 neurones artificiels composés chacun de 10 bascules, le système non protégé nécessite 740 bascules, tandis qu'une protection totale de toutes les bascules par triplication classique comme dans les applications avioniques nécessiterait 2220 bascules. La méthode proposée réduit considérablement le nombre de bascules nécessaires à 1200 pour le modèle ayant subi 1000 itérations d'apprentissage, soit une diminution de 45% par rapport à la méthode de triplication classique, tout en offrant une protection similaire.

5 Conclusion

Cet article a présenté un réseau de neurones implémenté sur FPGA en utilisant le langage VHDL. Un scénario de test a été développé pour détecter les bascules sensibles et une solution matérielle a été proposée pour la mise en œuvre de ce scénario. Les résultats obtenus montrent que les vulnérabilités aux fautes des couches d'un réseau de neurones ne sont pas identiques, la dernière couche est la plus sensible. De plus, il a été constaté que l'apprentissage joue un rôle important dans l'amélioration de la tolérance aux fautes intrinsèques du réseau de neurones. Augmenter l'apprentissage permet d'améliorer la tolérance aux fautes de l'architecture. Les images d'entraînement provoquent moins de fautes que les images de validation. Enfin, la méthode proposée dans cet article permet de réduire considérablement le coût matériel de la protection du modèle de réseau de neurones tout en offrant une protection similaire à une triplication classique.

D'autres travaux sont envisagés pour étudier l'impact de l'encodage et de la phase d'apprentissage sur la tolérance aux fautes d'un réseau de neurones.

6 Remerciements

Ce travail est soutenu par la Direction Générale de l'Armement (DGA), financé par l'Agence de l'Innovation de Défenses (AID) et l'institut national de recherche en sciences et technologies du numérique Inria.

Références

[1] Paul E Dodd and Lloyd W Massengill. Basic mechanisms and modeling of single-event upset in digital microelectronics. *IEEE Transactions on nuclear Science*, 50(3) :583–602, 2003.

[2] Melanie Berg and Kenneth A LaBel. Verification of triple modular redundancy (tmr) insertion for reliable and trusted systems. In *2016 MRQW Microelectronics Reliability and Qualification Working Meeting*, number GSFC-E-DAA-TN29375, 2016.

[3] Kiyohiro Furutani, Kazutami Arimoto, Hiroshi Miyamoto, Toshifumi Kobayashi, K Yasuda, and K Mashiko. A built-in hamming code ecc circuit for drams. *IEEE Journal of Solid-State Circuits*, 24(1) :50–56, 1989.

[4] Robert Chien. Cyclic decoding procedures for bose-chaudhuri-hocquenghem codes. *IEEE Transactions on information theory*, 10(4) :357–363, 1964.

[5] Alberto Bosio, Paolo Bernardi, Annachiara Ruospo, and Ernesto Sanchez. A reliability analysis of a deep neural network. In *2019 IEEE Latin American Test Symposium (LATS)*, pages 1–6, 2019.

[6] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15) :2627–2636, 1998.

[7] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[8] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6) :141–142, 2012.

[9] Isidoro Urriza, Luis Angel Barragan, Denis Navarro, José Ignacio Artigas, and Oscar Lucia. Word length selection method for controller implementation on fpgas using the vhdl-2008 fixed-point and floating-point packages. *EURASIP Journal on Embedded Systems*, 2010 :1–11, 2010.