PROJET GSE 2020-2021

Embedded AI on FPGA using High Level Synthesis

Groupe 2 :

GUO Ran

BARRIGA Ricardo

ELEC 5

Systèmes Embarqués

Tuteur de projet :

Sébastien BILAVARN

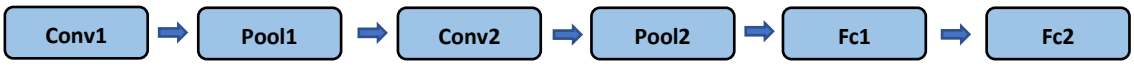
Sommaire

1.- Introduction :

L'objectif de ce projet est de développer un réseau neuronal convolutif accélérée sur une carte FPGA(Zedboard) en utilisant High Level Sysnthesis. Ce système va permettre de reconnaitre un chiffre entre 0 et 9. Ce projet a comme but aussi d'obtenir un temps d'exécution minimal et analyser l'Energie consommé.

2.-Developpement du projet Partie Software:

Nous avons passé pour différentes étapes pour réussir ce projet. D'abord on a dû générer le fichier hdf5 qui contient les pois de valeurs lors de l'apprentissage. Ensuite on doit utiliser ce fichier dans les fonctions de convolution et full connecting. Ensuite on adapte le code pour qu'il soit compatible avec HLS. Nous utilisons des pragmas pour augmenter l'utilisations des ressources de la carte. Nous accélérons les fonctions et nous avons testé sur la carte qu'on obtient une accélération. Nous allons décrire dans la suite ces différents étapes.

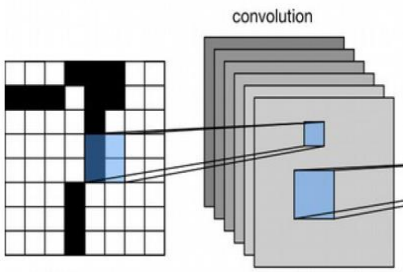


2.1 Generation du fichier hdf5 :

D'abord on a utilisé python pour générer le fichier hdf5 qui contient les pois à utiliser pour identifier le chiffre. Nous avons utilisé les librairies Keras/ TensorFlow pour compiler le code fourni. Ce fichier hdf5 contient les pois à utiliser dans les fonctions conv1, conv2, fc1, fc2. Puisque la carte Zedboard n'a pas la bibliothèque nécessaire pour lire ce fichier nous devons convertir ce fichier dans un fichier qu'on appelle weight.h.

2.2 Fonction conv1 :

Dans cette fonction nous allons parcourir des fenêtres de 5x5 avec les images de 28x28. Ces fenêtres sont des pois qui se trouvent dans le fichier weight.h et qui sont des filtres qui contient différentes caractéristiques des images des chiffres du 0 à 9. Nous utilisons un stride de 1 et chaque convolution va être enregistré comme une nouvelle image. Vu qu'on a dans le kernel 20 filtres nous allons générer en sortie un couche de 20 images.



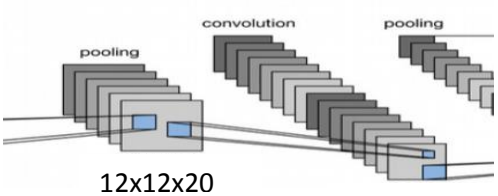
28x28x1 24x24x20

2.3 Fonction pool1 :

Cette fonction sert à réduire la complexité du réseau neuronal. Dans cette fonction nous allons garder la valeur la plus significative d'un bloc 2x2. Dans notre cas, nous allons garder la valeur plus haute (0 est noir et Maxval est blanc). Dans cette fonction on va recevoir en entrée l'output de conv1(24x24x20) et nous allons obtenir un output de 12x12x20. Nous pouvons noter qu'on garde toujours la profondeur de 20 et on a diminué la taille de chaque image de la couche. Aussi on s'assuré de forcer la valeur à 0 si la valeur choisie est négative.

2.4 Fonction Conv2 :

Cette deuxième convolution est un peu différente que la première. La principale raison est que maintenant on a en entrée plusieurs images, et donc une profondeur de 20. Cette fois-ci nous allons recevoir comme input 20 images de 12x12 et en sortie nous allons générer 40 images de 8x8. Cela parce que nous avons 40 filtres dans le fichier weight.h. Chacun de ces filtres de dimension 5x5x20. Dans cette fonction nous utilisons aussi un stride de 1.

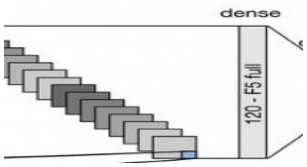


2.5 Fonction Pool2 :

Cette fonction est similaire à Pool1. Nous allons avoir comme entrée la sortie de la fonction conv2. Cela veut dire qu'on va avoir 40 images de 8x8 et en sortie nous allons obtenir 40 images de 4x4. Nous devons faire ça pour réduire le nombre de connexions qu'on génère et donc on garde les valeurs les plus importantes

2.6 Fonction fc1 :

Dans cette fonction nous allons connecter toutes les valeurs qu'on a obtenu précédemment. Donc on va additionner 400 fois tous les valeurs données par pool2 en les multipliant par les valeurs du kernel de fc1. De cette façon nous allons obtenir en sortie 1x1x400.



2.7 Fonction fc2 :

Dans cette fonction nous allons recevoir la sortie de fc1 et en sortie nous allons générer un tableau de 10 valeurs qui correspond à chaque chiffre entre 0 et 9. Chaque étiquette est connectée aux différents 400 valeurs de fc1 en utilisant le kernel de fc2. Donc pour chaque étiquette nous allons obtenir une addition diffèrent et un résultat significatif qui représente le chiffre identifié. Après en utilisant la fonction softmax nous allons pouvoir obtenir un résultat en pourcentage.

3.-Developpement du projet Partie Hardware :

Pour tester notre programme nous analysons 10000 images de taille 28x28 pixels en format .pgm. Après nous allons accélérer six principales fonctions : conv1, pool1, conv2, pool2, fc1, fc2. C'est pour ça que ce code doit être supporte par l'outil HLS(Xilinx Vivado HLS tool).

Pour le faire nous avons utilisé un changement de virgule flottant à virgule fixe. Aussi nous avons utilisé des boucles déterministes, des tableaux de taille statique, entre autres règles. Aussi nous avons incrémenté l'utilisation des ressources de la FPGA (slices, LUTs, BRAMs, DSP blocks) en appliquant des pragmas pour faire le pipelining, unrolling, entre autres.

Nous vérifions qu'on obtient une latence minimale et qu'on a respecté le temps minimal d'un cycle d'horloge pour que le système fonctionne. Après avoir obtenu un temps de latence acceptable nous utilisons SDSoC. Ce logiciel permet de générer un accélérateur de harware(RTL) en VHDL à partir de notre code en C. Cela va permettre utiliser différentes ressources de la carte et diminuer le temps d'exécution. Ce logiciel produit un exécutable que nous pouvons copier dans la micro SD et exécuter sur la carte Zedboard.

4.- Test and résultats :

4.1 Test de latence et timing :

Nous avons utilisé d'abord le logiciel Vivado pour vérifier qu'on a une latitude minimale. Nous avons obtenu le résultat final ci-dessous en utilisant des pragmas.

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.634	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
257120	257120	257120	257120	none

Detail

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	261
FIFO	-	-	-	-
Instance	121	47	8373	13515
Memory	34	-	485	161
Multiplexer	-	-	-	1454
Register	0	-	408	128
Total	155	48	9266	15519
Available	280	220	106400	53200
Utilization (%)	55	21	8	29

Nous pouvons voir qu'on a obtenu 245402 cycles de latence maximal et un timing estimé de 9.634 ns.

4.2 Test de temps d'exécution du programme sur le processeur :

Dans cette partie nous avons exécuté le programme sur la carte sans utiliser l'accélération des fonctions. Cela veut dire qu'on va faire tourner le programme que dans le processeur de la carte Zedboard.

```
Processing
TOTAL PROCESSING TIME (gettimeofday): 540.000000 s

Errors : 208 / 10000
Success rate = 97.919998%
Thw_min = 5789666 cpu cycles      Thw_max = 14463796 cpu cycles
root@zed:/media#
```

Nous pouvons voir dans ce résultat que notre programme dure 540s pour s'exécuter. Nous avons mesure aussi la quantité de cycles utilisés seulement dans la fonction lenet pour voir plus précisément l'accélération. Donc nous avons obtenu un maximal de 14463796 cycles.

4.3 Test de temps d'exécution avec accélération mais sans pragmas :

Dans ce test nous allons accélérer les fonctions, mais sans appliquer des pragmas, pour comparer dans la suite avec l'utilisation des pragmas.

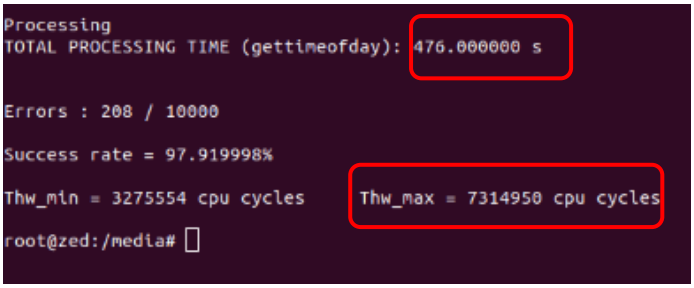
```
Reading weights
Opening labels file
Processing
TOTAL PROCESSING TIME (gettimeofday): 1115.000000 s

Errors : 208 / 10000
Success rate = 97.919998%
Thw_min = 43107494 cpu cycles     Thw_max = 47261974 cpu cycles
root@zed:/media#
```

Nous pouvons voir qu'on a obtenu 1115s de temps d'exécution et 47261974 cycles pour la fonction lenet.

4.4 Test de temps d'exécution avec accélération mais avec pragmas :

Dans ce test nous allons accélérer les fonctions, en appliquant des pragmas.



Nous pouvons voir qu'on a obtenu 476s de temps d'exécution et 7314950 cycles pour la fonction lenet. Nous pouvons remarquer que le temps d'exécution a diminué par rapport au test ou on a exécuté le programme que dans le processeur. Aussi on remarque qu'on a diminué de 47 millions à 7 millions par rapport au test sans pragma.

4.5 Test consommation d'énergie

D'abord nous avons analysé la consommation énergétique de notre programme qui s'exécute que dans le processeur. Nous avons trouvé 1.668 W (Voir Annexe 2). Donc l'énergie consommé est de 900.72Ws. Cette consommation énergétique est assez grande car le processeur est très complexe et grand. Nous cherchons diminuer cette consommation en utilisant HLS.

Ensuite nous avons analysé le programme sur le processeur sans pragma. Nous avons obtenu 0.178w. Cela parce qu'on n'utilise pas tous les ressources puisqu'on n'a pas appliqué des pragmas.

Ensuite nous avons analysé la consommation énergétique en utilisant pragmas. Nous pouvons voir dans les images en Annexe 2 qu'on a obtenu une consommation de 0.281 W et donc une énergie de 133.756 w, qui comparé à 900.72Ws(exécution que dans le processeur) nous pouvons remarquer une grande amélioration d'un factor de 10 environ dans la consommation énergétique. Nous pouvons noter qu'on a augmenté l'utilisation des ressources de la cartes tels que Luts, Lutram, BRAM, entre autres. C'est pour cela que nous obtenons une consommation plus haute que dans le cas de l'exécution sans pragma.

5.- Conclusion :

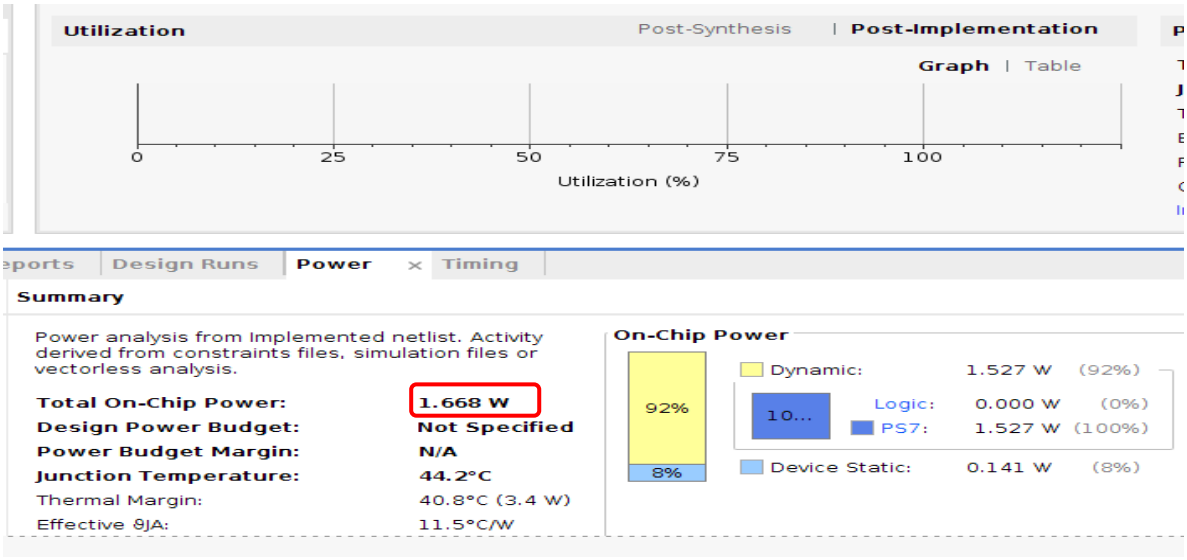
Ce projet a été très intéressant. Cela nous a aidé à découvrir la théorie de l'IA et mettre en place les solutions pour créer le CNN. Ce projet nous a permis concevoir tout un programme orienté pour l'embarqué en utilisant un FPGA. Nous avons remarqué qu'il est très important de vérifier et étudier la consommation énergétique et nous avons remarqué toutes les contraintes qu'il y a quand on veut faire un projet dans l'embarqué. Nous avons compris que le fait d'utiliser HLS aide a diminuer la consommation énergétique et améliorer le temps d'exécution, qui sont deux facteurs très importantes pour les systèmes embarqués.

Annexe 1 : Link Github du projet

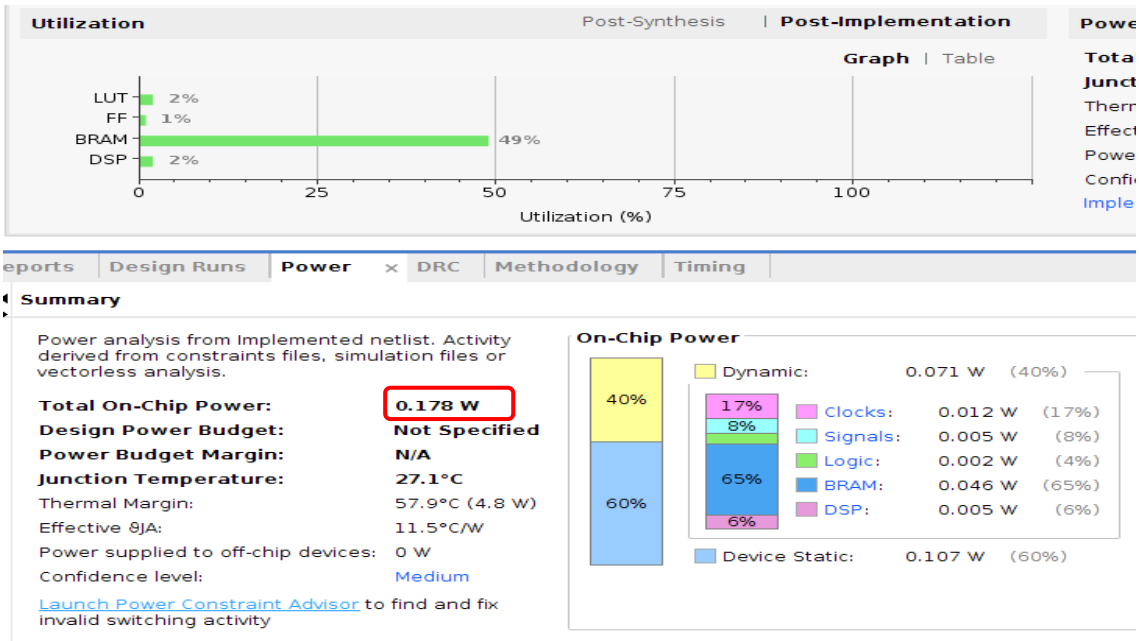
https://gitlab.polytech.unice.fr/br804273/projet_gse_guo_barriga_2020_2021

Annexe 2 : Consommation énergétique

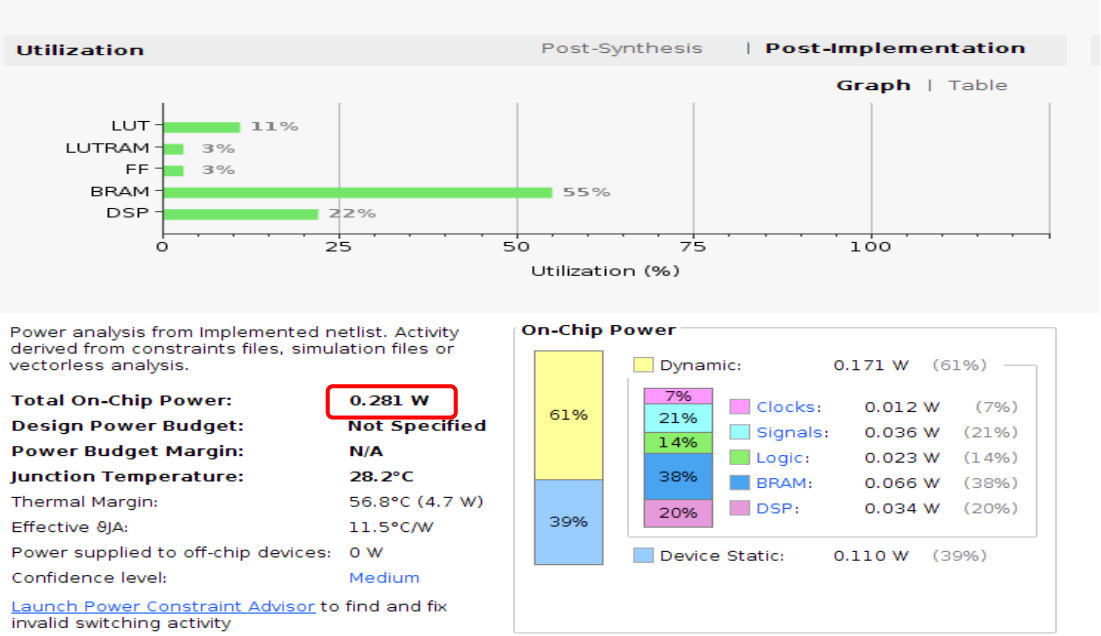
[1] Code sur processeur



[2] Code sur FPGA sans pragmas



[3] Code sur FPGA avec pragmas



Références :

LeNet-5 – A Classic CNN Architecture, https://engmrk.com/lenet-5-a-classic-cnn-architecture/

Convolutional Neural Networks (CNNs / ConvNets), https://cs231n.github.io/convolutionalnetworks/

convolutional_neural_networks.html

Looking inside neural nets, https://ml4a.github.io/ml4a/looking_inside_neural_nets/

Ubuntu 18.04: Install TensorFlow and Keras for Deep Learning,

https://www.pyimagesearch.com/2019/01/30/ubuntu-18-04-install-tensorflow-and-keras-for-deeplearning/

Coding Considerations (Xilinx),

http://home.mit.bme.hu/~szanto/education/vimima15/heterogen_vivado_hls_6.pdf

HLS pragmas (xilinx), https://www.xilinx.com/html_docs/xilinx2019_1/sdaccel_doc/hlspragmas-okr1504034364623.html e sur FPGA avec pragma