POLYTECH NICE SOPHIA

UNIVERSITÉ CÔTE D'AZUR

RAPPORT DE PROJET EMBEDDED AI ON FPGA USING HLS

DÉPLOIEMENT ET ACCÉLÉRATION D'UN CNN LENET

Encadrant :
Sébastien Bilavarn

Etudiantes
Malak MOURADI
Ibtissam EL FARJI

SOMMAIRE

Ibtissam EL FARJI
Malak MOURADI

1. INTRODUCTION

1.1 Contexte

Le projet vise l'accélération matérielle d'un CNN classique, LeNet-5 (reconnaissance de chiffres manuscrits MNIST), sur une plateforme Zynq-7000 (ZedBoard).

Le cœur de notre travail est d'utiliser la Synthèse Haut Niveau (HLS) (via Vivado HLS et l'environnement SDSoC/Vitis) pour transformer le code C/C++ du CNN en un accélérateur matériel (IP) dans la logique programmable du FPGA, exploitant ainsi la parallélisation intrinsèque de la PL.

1.2 Objectifs Détaillés

L'objectif principal est d'étudier le flot complet d'un modèle initial en virgule flottante vers une version embarquée optimisée en arithmétique fixe. Ce livrable est centré sur la comparaison de trois versions distinctes de l'application :

1. **SW (Software) :** Exécution séquentielle de l'inférence sur le processeur ARM Cortex-A9. Cette version sert de référence fonctionnelle et temporelle.
2. **HW_SEQ (Hardware Séquentiel) :** Accélération HLS de la fonction top-level du CNN, sans application des pragmas d'optimisation. Cette version permet d'évaluer l'impact des transferts de données CPU-FPGA.
3. **HW_PAR (Hardware Parallélisé) :** Accélération HLS utilisant des pragmas.

Ce projet s'inscrit également dans un objectif pédagogique visant à comprendre les compromis performance / ressources / complexité induits par l'accélération matérielle via HLS sur FPGA.

1.3 Architecture du CNN LeNet-5 implémenté

Le réseau de neurones implémenté est une adaptation du LeNet-5 pour la classification de MNIST:

Couche	Entrée (Dimensions)	Noyau/Filtres	Sortie (Dimensions)
Input	28 * 28 * 1	-	28* 28 * 1
Conv1	28 * 28 *1	20 * (5 * 5)	24 * 24 * 20
Pool1	24* 24 * 20	MaxPool 2 * 2 (Stride 2)	12 * 12 * 20
Conv2	12 *12 * 20	40 * (5 *5)	8 * 8 * 40
Pool2	8* 8* 40	MaxPool 2 * 2 (Stride 2)	4 * 4 *40
FC1	4 *4 *40 (Flatten to 640)	640 to 400 neurones	400
FC2	400	400 to 10 neurones	10
Output	10	Softmax	10 (Probabilités)

2. MÉTHODOLOGIE, OUTILS ET FLOT DE CONCEPTION

2.1 Flot de Conception Global

Le projet a suivi un flot en quatre étapes :

1. **Validation Logicielle FLOAT :**
   a. Le modèle est initialement entraîné (Keras/TensorFlow) et les poids sont stockés en virgule flottante (float32) dans un fichier HDF5 (lenet_weights.hdf5).
   b. Un code C de référence est écrit pour lire ces poids via des API HDF5 (utilitaires comme ReadConv1Weights_float dans utils.c) cf annexes fig1 et valider l'exactitude fonctionnelle (taux de succès MNIST approx 97.74%) cf annexes fig5.
2. **Conversion vers Fixed-Point :**
   a. Les API HDF5 et le format float sont incompatibles avec la synthèse HLS efficace.
   b. Le modèle est converti en arithmétique fixed-point (Q8 sur 16 bits). Les poids et les biais sont stockés en dur dans le fichier Weights.h (sous forme de tableaux static short const) cf annexes fig2
3. **Synthèse HLS et Génération d'Accélérateur :**
   a. La fonction top-level lenet_cnn_fixed est soumise à Vivado HLS.
4. **Construction et Déploiement du Système :**
   a. SDSoC orchestre la création du lien AXI (DMA) entre le PS et l'IP (PL).
   b. L'application (.elf) est compilée. Le système de boot (Linux) est généré.
   c. Les fichiers de l'application et du *boot* sont copiés sur la carte SD de la ZedBoard pour l'exécution finale.

2.2 Environnement matériel et logiciel

Les expérimentations ont été réalisées sur une plateforme Zynq-7000 (ZedBoard), intégrant un processeur ARM Cortex-A9 (667 MHz) et une logique programmable FPGA.

Le flot de conception s'appuie sur :

- Vivado HLS 2018.3 pour la synthèse haut niveau

-SDSoC 2018.3 pour l'intégration système (PS/PL, AXI, DMA)

- Linux embarqué généré automatiquement par SDSoC

- Compteurs Xilinx (sds_clock_counter) pour la mesure des temps matériels

3. IMPLÉMENTATION ET VALIDATION ARITHMÉTIQUE FIXED-POINT

3.1 Conversion du Modèle Float vers l'Arithmétique Q8 (16 bits)

Nous avons opté pour le format Q8 cf annexes fig3. Ce format utilise des entiers 16 bits (short), répartis comme suit :

- 1 bit de signe (S).
- 7 bits pour la partie entière (I).
- 8 bits pour la partie fractionnaire (F).

La conversion d'une valeur réelle x en sa représentation fixe x_fixed s'effectue par l'opération de mise à l'échelle :

$$x\_Fixed = round(x * 2^8)$$

Cette conversion a été réalisée :

1. Les poids FLOAT sont lus depuis le fichier HDF5 sur PC.
2. Ils sont convertis en short Q8.
3. Ces valeurs fixes sont intégrées en dur sous forme de tableaux static short const dans le fichier Weights.h.

Le format Q8 a été retenu comme compromis entre précision numérique et complexité matérielle, permettant de préserver les performances de classification tout en facilitant la synthèse HLS efficace.

3.2 Adaptation des Calculs (MAC, Bias, Scaling)

Les données d'entrée, les activations et les poids sont tous stockés en short (Q8). La multiplication de deux nombres Q8 produit un nombre au format Q16.

1. Accumulation Sécurisée
2. Alignement du Biais
3. Opération MAC
4. Renormalisation (Scaling)

Cette logique de calcul est visible dans les fonctions de convolution (conv_fixed.c) et de couches fully-connected (fc_fixed.c) cf annexes fig 4. Le résultat final passe ensuite par l'activation ReLU avant d'être stocké dans un short.

4. CONCEPTION MATÉRIELLE

4.1 Stratégies d'Optimisation (HW_SEQ vs HW_PAR)

HW_SEQ (Séquentiel)

Nous laissons le code s'exécuter séquentiellement (comme sur un CPU, mais avec des composants FPGA).

- **Latence (Temps) : Élevée** (lent), car les calculs sont faits un par un.
- **Ressources FPGA : Faible** consommation, car l'outil n'a pas besoin de dupliquer beaucoup de circuits de calcul.

cf annexes fig 6 et 7.

HW_PAR (Parallélisation Maximale)

Nous forçons le compilateur HLS à faire un maximum de calculs en même temps (parallélisation).

- **Latence (Temps) : Faible** (très rapide), car de nombreuses opérations MAC s'exécutent en parallèle.
- **Ressources FPGA : Très Élevée** consommation, car :
    - Elle utilise beaucoup de blocs **DSP** (pour les multiplications parallèles).
    - Elle utilise beaucoup de blocs **LUT** et **FF** (pour la logique de contrôle complexe nécessaire au parallélisme).

5. VALIDATION FONCTIONNELLE ET ANALYSE DES PERFORMANCES

5.1 Déploiement sur ZedBoard et Diagnostics

Après la compilation par SDSoC, les fichiers suivants sont copiés sur la carte SD, montée sous Linux embarqué : BOOT.BIN, image.ub, et l'exécutable .elf

Les performances ont été mesurées selon deux métriques :

- Le nombre de cycles matériels issu des rapports Vivado HLS

- Le temps physique d'exécution mesuré sur la plateforme embarquée

à l'aide des compteurs Xilinx (sds_clock_counter)

Les mesures correspondent au traitement complet d'une imagette MNIST.

5.2 Résultats Comparatifs et Analyse Ressources/Performance

Version	Taux de Succès	Tglobal (Total, s)	Hw Cycles
SW	97.74%	2400	N/A
HW_SEQ	97.74%	2700	6 447 885
HW_PAR	97.74%	300	750 235

Analyse

- **Observation (HW_SEQ)** : Il est fréquent que la version HW_SEQ soit *plus lente* que la version SW. Cela s'explique par l'overhead des transferts de données (AXI) entre la mémoire PS (DDR) et l'accélérateur PL. Si l'accélérateur ne gagne pas suffisamment de temps par son exécution interne, le temps de transfert domine.

- **Observation (HW_PAR)** : La version HW_PAR démontre une accélération significative. Le gain obtenu par la réduction drastique de la latence HLS est suffisant pour masquer l'overhead des transferts, ce qui prouve l'efficacité de la parallélisation.

6. CONCLUSION ET PERSPECTIVES

Ce projet a permis de réaliser avec succès le cycle de conception complet d'une intelligence artificielle embarquée, en commençant par la transition réussie du modèle flottant vers un format Fixed-Point Q8 synthétisable sans perte de précision notable. Malgré des défis complexes de Robustesse HLS , le code a été optimisé pour la synthèse matérielle. L'application ciblée des directives HLS a permis d'obtenir une Accélération Matérielle significative des opérations clés de Multiplication-Accumulation (MAC), avec un gain de performance. Finalement, le Déploiement Complet sur la carte ZedBoard via le flot SDSoC/Linux a confirmé l'opérationnalité et l'efficacité de l'ensemble du système hétérogène CPU/FPGA.

Des perspectives d'amélioration incluent :

- L'exploration de formats Fixed-Point plus agressifs (Q6, Q4)

- L'optimisation des transferts mémoire (streaming, double buffering)

- L'utilisation d'architectures CNN plus modernes

Annexes

Fig1 : fonctions dans utils.c

```
void ReadConv1Weights_float(char *filename, char *datasetname,
                            float W[CONV1_DIM][CONV1_DIM][IMG_DEPTH][CONV1_NBOUTPUT])
{
    hid_t file, dataset;
    herr_t status;

    file    = H5Fopen(filename, H5F_ACC_RDONLY, H5P_DEFAULT);
    dataset = H5Dopen(file, datasetname, H5P_DEFAULT);

    status = H5Dread(dataset, H5T_NATIVE_FLOAT,
                     H5S_ALL, H5S_ALL, H5P_DEFAULT, W);

    H5Dclose(dataset);
    H5Fclose(file);
}
```

Fig2 : extrait de weights.h

```
static short CONV1_KERNEL[CONV1_NBOUTPUT][IMG_DEPTH][CONV1_DIM][CONV1_DIM] =
{
{{{0,16,33,3,-33,},{21,8,-13,1,-32,},{46,48,28,11,25,},{12,1,-18,17,12,},{30,0,-15,22,-8,},},},
{{{-8,-21,-19,-28,-13,},{0,-14,-26,4,47,},{9,-37,-19,23,24,},{-20,16,-19,40,34,},{-28,-24,1,60,33,},},},
{{{-37,-20,-32,-12,-6,},{-37,-5,0,30,12,},{8,9,25,43,0,},{31,30,12,-9,23,},{17,30,-18,15,5,},},},
{{{-8,-19,-9,1,5,},{-7,-19,6,-23,-12,},{6,-2,-19,14,8,},{-14,12,-7,-25,-15,},{-26,0,9,0,18,},},},
{{{-15,26,22,53,11,},{23,34,61,46,28,},{2,22,64,69,28,},{11,24,58,53,19,},{25,28,67,43,36,},},},
```

Fig3 : extrait de lenet_cnn_fixed_point.h

```
#define FIXED_POINT 8    // Q8
```

Fig 4 : extrait de cconv_fixed

```
void Conv1_28x28x1_5x5x20_1_0_fixed(
        short input[IMG_DEPTH][IMG_HEIGHT][IMG_WIDTH],
        short kernel[CONV1_NBOUTPUT][IMG_DEPTH][CONV1_DIM][CONV1_DIM],
        short bias[CONV1_NBOUTPUT],
        short output[CONV1_NBOUTPUT][CONV1_HEIGHT][CONV1_WIDTH])
{
    unsigned short k, z, y, x, ky, kx;
    for (k = 0; k < CONV1_NBOUTPUT; k++) {
        for (y = 0; y < CONV1_HEIGHT; y++) {
            for (x = 0; x < CONV1_WIDTH; x++) {
                /* Accumulate in 32-bit to avoid overflow */
                int acc = ((int)bias[k]) << FIXED_POINT;
                for (z = 0; z < IMG_DEPTH; z++) {
                    for (ky = 0; ky < CONV1_DIM; ky++) {
                        for (kx = 0; kx < CONV1_DIM; kx++) {
                            unsigned short in_y = (unsigned short)(y + ky);
                            unsigned short in_x = (unsigned short)(x + kx);
                            acc += ( (int)input[z][in_y][in_x] *
                                     (int)kernel[k][z][ky][kx] );
                        }
                    }
                }
                /* Return to FIXED range */
                acc >>= FIXED_POINT;
                output[k][y][x] = relu_fixed((short)acc);
            }
        }
    }
}
```

Fig 5 : Software avec fixed point

```
19    {{{-32,31,74,33,3,},{10,71,72,40,-30,},{64,81,78,-12,-48,},{
PROBLEMS  OUTPUT  TERMINAL  ...   powershell
 about_Command_Precedence".
PS C:\Users\ibtis\OneDrive\Bureau\Projet_bilavarn\HLS_IA\FIXED_POINT> ./test
_lenet

TEST FINISHED
Errors: 226 / 10000
Success rate: 97.74%
PS C:\Users\ibtis\OneDrive\Bureau\Projet_bilavarn\HLS_IA\FIXED_POINT>
0 0   Ln 9, Col 24  Spaces: 4  UTF-8  LF  { } C  Win32
```

Fig 6 : résultat en Hardware

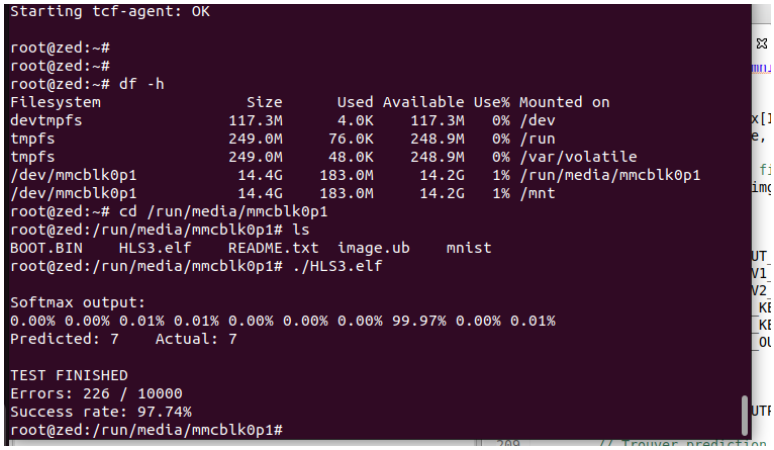

Fig 7 : Avant optimisation