الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

UNIVERSITÉ BADJI MOKHTAR - ANNABA
BADJI MOKHTAR – ANNABA UNIVERSITY

جامعة باجي مختار – عنابـــة

Faculté : Sciences de L'ingéniorat

Département : Electronique

Domaine : Sciences et Techniques

Filière : Electronique

Spécialité : Electronique des systèmes embarqués

Mémoire

Présenté en vue de l'obtention du Diplôme de Master

Thème:

Étude de l'implémentation d'un réseau de neurones convolutif CNN sur FPGA

Présenté par : *Yakoubi Dounia*

Benzine Asma

Encadrant : *Yahi Amira* *MCB* U.ANNABA

Jury de Soutenance :

BENOUARET Mohammed	PR	U.ANNABA	Président
YAHI Amira	MCB	U.ANNABA	Encadrant
ATTOUI Hamza	MAA	U.ANNABA	Examinateur

Année Universitaire : 2020/2021

Résumé

Le deep Learning ou apprentissage profond est un type d'intelligence artificielle dérivé de la machine Learning (apprentissage automatique). Il est capable de gérer une quantité de données énorme afin de donner aux machines la capacité d'apprendre par elle-même.

Dans notre travail, nous avons utilisé un modèle CNN (Convolutional Neural Network ou réseau de neurones convolutifs) qui est l'un des techniques les plus performants du deep Learning, dans le domaine de la reconnaissance et classification des images pour créer un algorithme qui permet de résoudre des équations écrites à la main. Ce modèle CNN a été entraîné, mis en œuvre en utilisant vivado (xilinx) puis exécuté sur une plateforme de type FPGA (PYNQ Z2) afin d'accélérer les calculs. Les étapes d'implémentation sont bien détaillées dans ce mémoire.

Remerciements

Notre remerciement s'adresse en premier lieu à Allah. Nous souhaitons avant tout remercier notre encadreur de mémoire, Mme YAHI AMIRA, pour le temps qu'elle a consacré et sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter notre réflexion.

Sans oublier de remercie les membres du jury qui nous font honneur en jugeant ce travail

Enfin, un grand merci à nos parents, pour leur amour et leurs conseils et nous tenons à exprimer notre gratitude à nos amis qui nous ont apporté un soutien moral et intellectuel, Tout au long de notre parcours.

Sommaire

Liste des figures

Liste des tableaux

Liste des acronymes

ANN: Artificial Neural Networks

ASIC: application-specific integrated circuit

CNN: Convolutional neural network

CUDA : Compute Unified Device Architecture

DL: Deep Learning

DMA : Direct Memory Access

DSP: Digital Signal Processor

FPGA: Field Programmable Gate Arrays

GPU: Graphics processing unit

HLS: High Level Synthesis

HMC: Handwritten Mathematical Calculator

IP: Intellectual property

MAC: Multiply and Accumulate

RTL: Register Transfer Level

SIMD : Single Instruction Multiple Data

TPU : Tensor Processing Unit

Chapitre 1: Introduction

Chapitre 1 : Introduction

1. Introduction générale :

Les CNN (Convolutional Neural Network ou réseau de neurones convolutifs en français) est un type de réseau de neurones artificiels acycliques [1], dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les neurones de cette région du cerveau sont arrangés de sort qu'ils correspondent à des régions qui se chevauchent lors du pavage du champ visuel. Leur fonctionnement est inspiré par les processus biologiques, ils consistent en un empilage multicouche de perceptron dont le but est de prétraités des petites quantités d'informations. Les réseaux neuronaux convolutifs ont de larges applications dans la reconnaissance d'image et vidéo, les systèmes de recommandation et le traitement du langage naturel.

Les premiers travaux sur les réseaux neuronaux convolutifs (CNN) modernes ont eu lieu dans les années 1990, inspirés par le neocognitron Yann LeCun et al, dans leur article "Gradient-Based learning Applied to Document Recognition" (aujourd'hui cité 17 588 fois) ont démontré qu'un modèle CNN qui regroupe des caractéristiques simples en caractéristiques progressivement plus complexes peut être utilisé avec succès pour la reconnaissance de caractères manuscrits. [2]

LeCun et al. A formé un CNN en utilisant la base de données du MNIST de chiffres manuscrits. C'est un ensemble qui comprend des images de chiffres manuscrits appariés avec leur véritable étiquette de 0, 1, 2, 3, 4, 5, 6, 7, 8 ou9, en lui demandant de prédire quel chiffre est montré dans l'image, puis en mettant à jour les paramètres du modèle selon qu'il a prédit l'identité du chiffre correctement ou non.

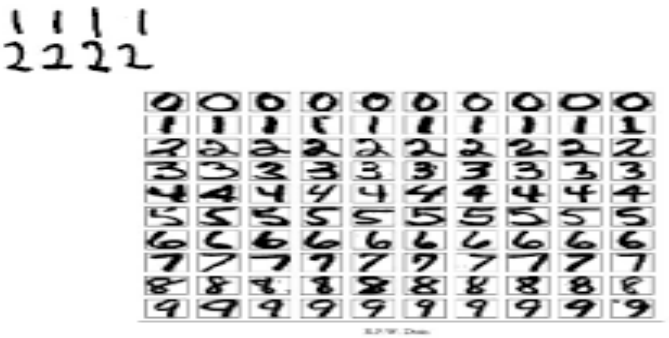


Figure 1.1 : Techniques de reconnaissance automatique de l'écriture manuscrite

Chapitre 1 : Introduction

La reconnaissance d'image : est un domaine scientifique multidisciplinaire qui traite la façon dont les ordinateurs utilisent l'analyse d'image ou des vidéos numérique. Du point de vue de l'ingénierie, la reconnaissance d'images vise à automatiser les taches que le système visuel humain peut effectuer.

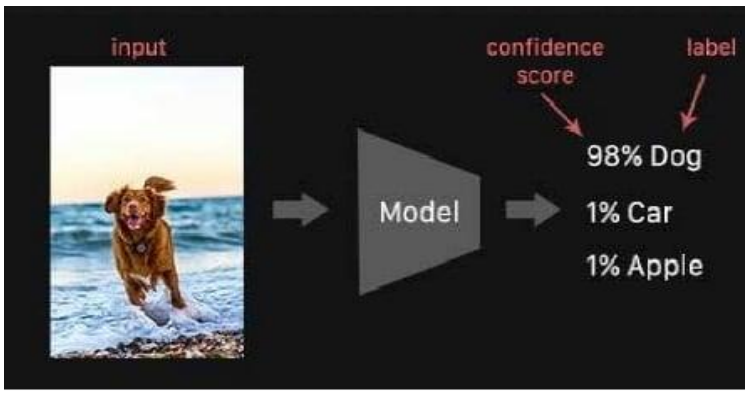


Figure 1.2 : Exemple de reconnaissance d'image.

Les CNN ont gagné en popularité grâce à leur capacité à effectuer la détection et la reconnaissance d'objets avec des performances humaines compétitives sur certaines tâches et une relative facilité de construction par rapport aux autres algorithmes. Pour détecter et reconnaitre correctement un objet, les CNN doivent effectuer des opérations intensives et des calculs avec numéros de points de rotation.

Les modèles CNN sont généralement entrainés et exécutés sur des unités de traitement graphique(GPU) car le GPU peuvent utiliser un parallélisme massif pour les calculs du nombre de points de rotation. Les alternatives aux GPU pourraient être les unités centrales de traitement (CPU), les matrices de portes programmables sur site (FPGA) et les circuits intégrés spécifiques aux applications (ASIC), à savoir les unités de traitement tensoriel (TPU). Chaque plate-forme matérielle a ses propres limites et avantages. Les applications de vision par ordinateur sur les plates-formes embarquées pourraient bénéficier de la précision des CNN.

Les (GPU) consommeraient trop d'énergie pour les applications embarquées, les processeurs ne seraient pas en mesure d'atteindre des performances en temps réel et les ASIC seraient couteux à concevoir et à fabriquer. Par rapport aux autres plates-formes matérielles, les FPGA peuvent offrir des solutions flexibles et efficacité énergétiques pour les applications CNN sur plates- formes embarquées. Les CNN peuvent être conçus dans des cadres tels que Theano

Tensor. Cependant, aucun des Framework n'offre de solutions qui pourraient être directement implémentées sur FPGA.

Afin de diminuer l'utilisation accrue de la mémoire due aux massifs calculs nécessaires pour la récentes CNN, des architectures matérielles ont été proposées par l'industrie et les communautés de recherche. Cependant, il existe trois chemins de conception principaux que l'on peut suivre lors du développement d'architectures matérielles pour les algorithmes d'apprentissage en profondeur. Le premier consiste à adopter l'utilisation des GPU, ou les développeurs implémentent un réseau en le programmant et tirant parti du haut niveau de parallélisme fourni.

En outre pendant cette phase, des opérations en virgule flottant de haute précision (également fournies par les GPU) sont requises par des algorithmes tels que la rétro propagation. Les deux autres chemins possibles incluent les FPGA ou les ASIC. Le premier est l'un des choix les plus flexibles et le secondes atteint les performances les plus élevées

2. Contribution du mémoire :

Dans notre travail, nous avons mis en œuvre un model CNN prédéfini qui consiste en une calculatrice mathématique manuscrite, et ceci sur une carte FPGA de type Pynq Z2 (Xilinx). Ce model utilise l'image stockée sur la carte SD de l'FPGA, détecte les nombres et les opérateurs mathématiques dans l'image, extrait ces éléments séparément en les redimensionnant en 32x32. Le réseau de Neurone convolutif produira par la suite 14 classes qui définissent respectivement les nombres de 0 à 9 puis les 4 opérations possibles à savoir : +, -, *, /. Finalement le résultat de l'opération est calculé puis affiché.

L'architecture du CNN prédéfini dispose de deux parties distinctes :

- Une partie logicielle : qui contient le code utilisé pour crée l'architecture du CNN en utilisant la bibliothèque du TENSORFLOW
- Une partie matérielle : qui contient le BLOC DESIGN qui va prédéterminer et configurer les ressources de la carte FPGA à utiliser pour implémenter l'architecture du CNN.

Ce mémoire est divisé en quatre chapitres :

Chapitre 1 : Introduction

Dans le premier chapitre nous présentons une introduction générale sur les CNN et la reconnaissance d'image avec quelque exemple de l'industrie. Le deuxième chapitre concerne l'étude des concepts des différentes phases d'apprentissage, ainsi que la constitution des réseaux de neurones. Dans le troisième chapitre, nous présentons l'architecture des CNN ainsi que ses différentes couches dans un FPGA. Finalement le quatrième chapitre détaillera les étapes d'implémentation du modèle CNN sur la plateforme FPGA, ainsi que les résultats obtenus.

Chapitre 2 : Les réseaux de neurones et CNN

Chapitre 2: Les réseaux de neurones et CNN

1. Introduction :

Depuis l'antiquité, le sujet de machines pensantes préoccupe les esprits. Ce concept est la base de pensées pour ce qui deviendra ensuite l'intelligence artificielle, ainsi qu'une de ses sous-branches : l'apprentissage automatique.

Dans l'apprentissage automatique, les taches sont généralement classées en grandes catégories basées sur la façon dont l'apprentissage est reçu, ou comment le feedback sur l'apprentissage est donné au système développé des deux méthodes d'apprentissage automatique les plus largement adoptées sont l'apprentissage supervisé et l'apprentissage non supervisé.

2. Le réseau neuronal :

Les réseaux neuronaux, aussi appelés ANN (Artificiel Neural Networks), est un système dont la conception est à l'origine schématiquement inspirée du fonctionnement des neurones biologiques, et qui par la suite s'est rapproché des méthodes statistiques [1].

2.1 Le neurone :

Ou une cellule nerveuse, est une cellule excitable constituant l'unité fonctionnelle de la base du système nerveux. Les neurones assurent la transmission d'un signal bioélectrique appelé influx nerveux. Les deux figures suivantes (figure 2.1 et figure 2.2) montrent une représentation d'un neurone réel et d'un neurone artificiel.

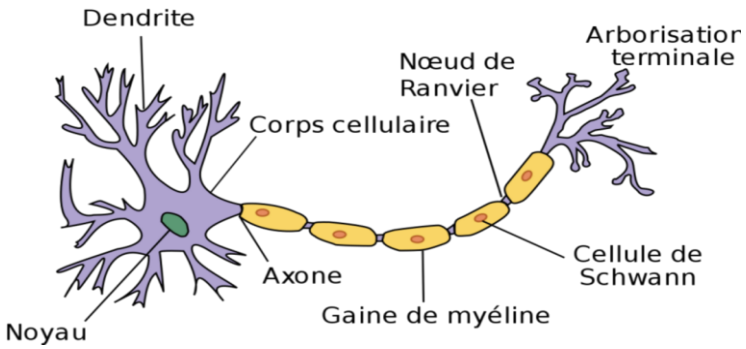


Figure 2.1 : Représentation d'un réseau de neurone

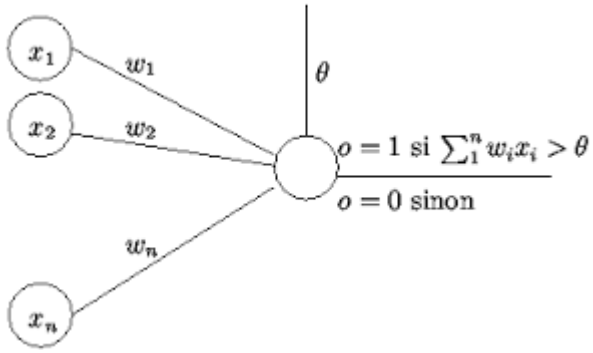


Figure 2.2 : Le modèle de neurone

Les X_1... X_n (sont des valeurs numériques qui représentent soit les données d'entrée comme une somme) sont les valeurs de l'information : dans le cas d'un ordinateur, il s'agira de 0 ou de1 (car toutes les informations sont en code binaire).

Cela correspond à l'information récupérée par les synapses. Les Wi sont les poids, soit les coefficients, correspondant aux Xi : ils permettent selon leur ajustement de donner la quantité. Le produit (Wi*Xi) est ensuite soumis à un test, pour savoir Si inferieure ou supérieure à un seuil fixé, permettant de décider la valeur qui sera transmise. Ainsi, le neurone peut transmettre une information issue de ses différents synapse (ou entrées). [19]

2.2. Les fonctions d'activation :

Dans le domaine des réseaux de neurones artificiels, la fonction d'activation est une fonction mathématique appliquée à un signal en sortie d'un neurone artificiel. La fonction d'activation est souvent une fonction non linéaire... un exemple de fonction d'activation est la fonction de Heaviside, qui renvoie tout le temps 1 si le signal en entrée est positif ou 0 s'il est négatif [2].

3. Les réseaux de neurones convolutionnés :

Les réseaux de neurones convolutionnels sont un type de réseau de neurones spécialisés pour le traitement de données. Sont à ce jour les modèles les plus performants pour classer des images. Désignés par l'acronyme CNN, de l'anglais Convolutional Neural Network [3].

Ce type de réseau est souvent constitue de trois types de couches qui sont la couche de convolution lie souvent avec la fonction d'activation RELU, la couche de pooling la couche de correction Relu et la couche entièrement connectée.

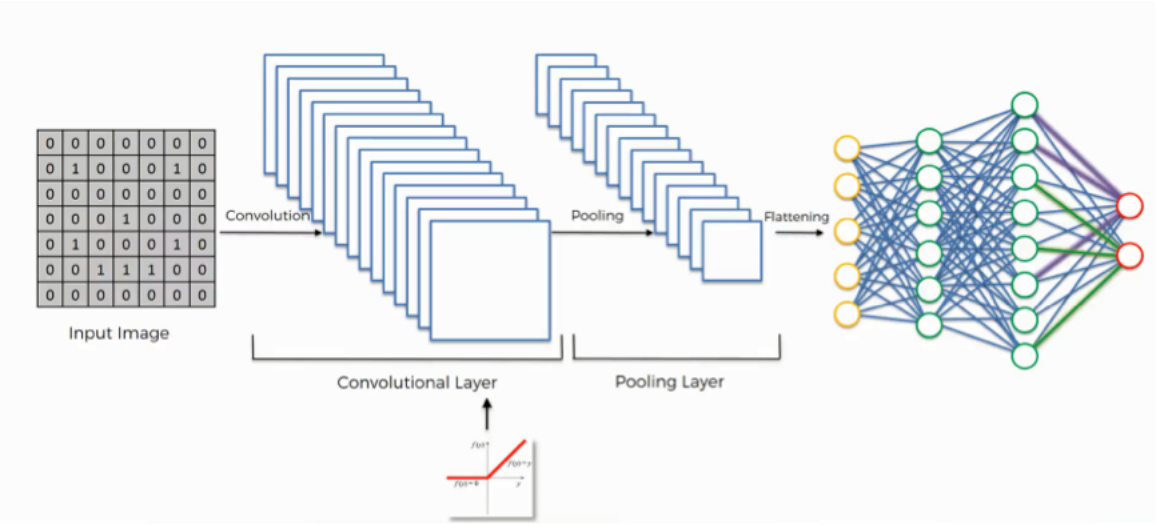


Figure 2.3 : vue générale de la couche de CNN

Ces couches effectuent des opérations qui modifient les données dans le but d'apprendre des fonctionnalités spécifiques aux données.

3.1. La Convolution :

La convolution est la partie la plus importante des CNN, fait passer les images d'entrée à travers un ensemble de filtres, dont chacun active certaines fonctionnalités des images. L'unité linéaire rectifiée (RELU) permet un entrainement plus efficace en mappant les valeurs négatives à zéro et en maintenant les valeurs positives. Ceci est parfois appelé activation, car seules les entités activées sont reportées dans la couche suivante.

La mise en commun simplifie la sortie en effectuant un sous- échantillonnage non linéaire, réduisant le nombre de paramètres que le réseau doit apprendre Ces opérations sont répétées sur des dizaines ou des centaines de couche apprenant à identifier différentes caractéristiques.

Son but, pour cela on réalise un filtrage par convolution : le principe est fait "glisser" le filtre sur l'image, et de calculer le produit de convolution entre le filtre et chaque portion de l'image balayée [2].

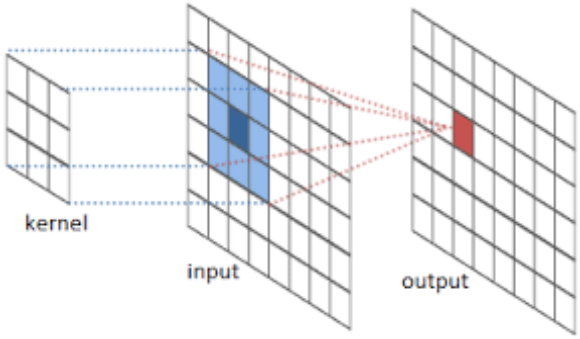


Figure 2.4 : l'utilisation des filtres pour obtenir des cartes d'activation

3.2. Le pooling:

Ce type de couche est souvent placé entre deux couche de convolution elle reçoit en entrée plusieurs feature maps et applique à chacun d'entre elles l'opération de pooling. Cette permet de réduire le nombre de paramètres et de calculs dans le réseau, On améliore ainsi l'efficacité du réseau et on évite le sur apprentissage.

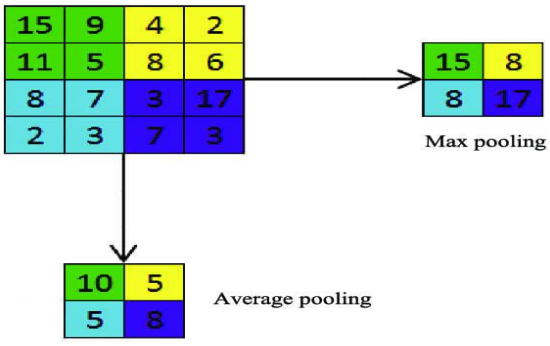


Figure 2.5 : Exemple Average versus max pooling.

L'opération consiste à découper l'image en cellules régulières, puis on garde au sein de chaque cellule la valeur maximale. En pratique, on utilise souvent des cellules carrées de petite taille

pour ne pas prendre trop d'informations. Les choix les plus communs sont des cellules adjacentes de taille 2*2 pixels qui ne se chevauchent pas ou des cellules de taille 3*3 pixels, distantes les unes des autres d'un pas de 2 pixels (qui se chevauchent donc) [4].

3.3. La fonction RELU :

RELU (Rectified Linear Unités) désigne la fonction réelle non-linéaire définie par RELU(x) = max (0, x). La couche de correction RELU remplace donc toutes les valeurs négatives reçues en entrées pas des zéros et garde les valeurs positives, elle joue le rôle de fonction d'activation [4].

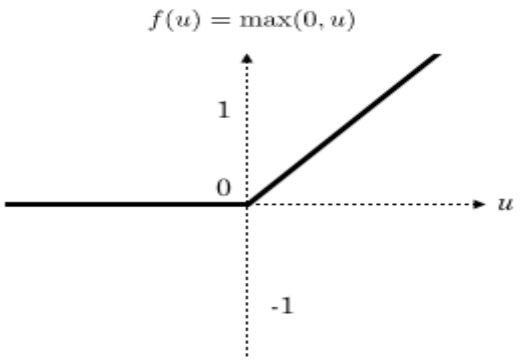


Figure 2.6 Allure de la fonction RELU

3.4. Fully -connected:

La couche fully-connected constitue toujours la dernière couche d'un réseau de neurones, convolutifs ou non elle n'est donc pas caractéristique d'un CNN. Ce type de couche reçoit un vecteur en entrée et produit un nouveau vecteur en sortie. Pour cela, elle applique une combinaison linéaire puis éventuellement une fonction d'activation aux valeurs reçues en entrée [2].

La dernière couche fully-connected permet de classifier l'image en entrés du réseau. Elle renvoie un vecteur de taille N, où N est le nombre de classe. Chaque élément du vecteur indique la probabilité pour l'image en entrée d'appartenir.

La fonction pour le calcul est donnée par :

$$f^{(l)}[n] = \text{act}\left[b^{(l)}[n] + \sum_{c=1}^{C^{(l)}} < \phi^{(l)}[c], w^{(l)}[n,c] >\right]$$

Par exemple, si le problème consiste à distinguer les chats des chiens, le vecteur final sera de taille 2. Le premier élément donne la probabilité d'appartenir au classe "chat", le deuxième a la classe "chien". Ainsi, le vecteur [0.9 0.1] signifie que l'image à 90% de chances de représenter un chat. Chaque valeur du tableau en entrée "vote" en faveur d'une classe. Les votes n'ont pas tous la même importance : la couche leurs accorde des poids qui dépendent de l'élément du tableau et de classe.

La fonction Softmax:

Pour calculer les probabilités, la couche fully_connected multiplie donc chaque élément en entrée par un poids, fait la somme, puis applique une fonction d'activation (logistique si N = 2, softmax si N > 2) :

La fonction est définie par :

$$\sigma(\mathbf{z})_j = \frac{\mathbf{e}^{z_j}}{\sum_{k=1}^{K} \mathbf{e}^{z_k}} \text{ pour tout } j \in \{1, \ldots, K\}.$$

Ce traitement revient à multiplier le vecteur en entrée par la matrice contenant les poids. Le fait que chaque valeur en entrée soit connectée avec toutes les valeurs en sortie explique le terme fully_connected [2].

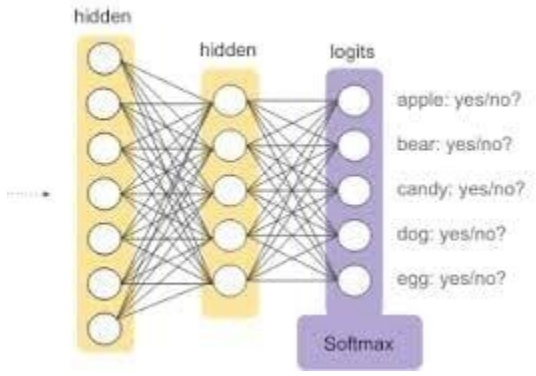


Figure 2.7 : la fonction soft max.

4. L'entrainement d'un réseau de neurones

L'entraînement d'un réseau de neurones [5] se compose généralement de deux phases :

- Une phase **avant**, où l'entrée est entièrement passée à travers le réseau.
- Une phase **arrière**, où les gradients sont rétropropagés (backprop) et les poids sont mis à jour.

Pendant la phase avant, chaque couche mettra en cache toutes les données (telles que les entrées, les valeurs intermédiaires, etc.) dont elle aura besoin pour la phase arrière. Cela signifie que toute phase arrière doit être précédée d'une phase avant correspondante.

Pendant la phase de retour, chaque couche recevra un dégradé et renverra également un dégradé. Il recevra le gradient de perte par rapport à ses *sorties* ($\partial L / \partial out$) et renverra le gradient de perte par rapport à ses *entrées* ($\partial L / \partial in$).

5. Framework :

Les modèles CNN peuvent être conçus dans des Framework comme Caffe, Theano, Tensorflow, DarkNet, etc. Les Framework fournissent des fonctionnalités permettant de construire des couches pour les modèles, d'entrainer les modèles et exécuter l'inférence (exécuter un modèle sans rétropropogation) [6]. Les Framework sont construits pour utiliser les GPU et les CPU pour effectuer l'entrainement et l'inférence. Selon la configuration CUDA utilisée par les Framework, plusieurs GPU_s pourraient utilises en parallèle pour l'entrainement, par exemple CAFFE supporte plusieurs GPU mais TENSORFLOW et THEANO ne peuvent fonctionner qu'avec un seul.

A cause des ressources limites des $FPGA_s$ et la complexité de la retropropagation, effectuer l'entrainement complet est très difficile. Malheureusement, aucun framework n'offre de solutions qui pourraient être directement implémentées sur un FPGA, et donc des applications tierces doivent être utilisées pour exécuter l'inférence sur des FPGA.

6. Hardware :

Pour détecter et reconnaître correctement un objet, les CNN doivent effectuer des opérations de multiplication et d'accumulation (MAC) avec des nombres à virgule flottante. Les réseaux sont généralement formés et exécutés sur des unités de traitement graphique (GPU), car les GPU peuvent utiliser un parallélisme massif pour les calculs de nombres à virgule flottante. L'exécution de l'inférence des réseaux neuronaux nécessite beaucoup moins de calculs. En plus des GPU, l'inférence peut être exécutée sur des CPU, des FPGA et des ASIC, notamment des TPU [6]. Une brève description des plates-formes matérielles utilisées pour les CNN est présentée ci-dessous :

- Les CPU peuvent exécuter des instructions pour effectuer les calculs arithmétiques nécessaires à l'inférence. Bien que les CPU puissent utiliser des instructions SIMD et des cœurs multiples pour exécuter plusieurs calculs arithmétiques en parallèle, ils n'auraient toujours pas la vitesse par rapport aux autres alternatives.
- Les GPU sont des processeurs à plusieurs cœurs conçus pour traiter des blocs de données en parallèle. CUDA et OpenGL sont des frameworks qui peuvent être utilisés pour distribuer des données sur plusieurs processeurs et sur la mémoire interne d'un GPU. Les GPU peuvent stocker jusqu'à 12 Go de données dans une mémoire à large bande passante, ce qui permet un accès rapide aux données en virgule flottante. Cependant, l'immense puissance de traitement se paie au prix de la consommation d'énergie des GPU qui peut atteindre jusqu'à 250W.
- Les FPGA offrent des architectures matérielles flexibles, car différentes unités de calcul peuvent être utilisées, contrairement aux processeurs ou aux processeurs graphiques qui sont fixés à une seule architecture. Les FPGA peuvent offrir des solutions flexibles qui

  S’adaptent au rythme des développements rapides de la recherche sur les réseaux neuronaux. Ces couches peuvent être placées sur un FPGA et exécutées en parallèle pour une inférence efficace.
- - Les TPU sont des ASIC conçus pour réaliser l'inférence de réseaux neuronaux et améliorer les performances en termes de coûts par rapport aux GPU. Les TPU sont optimisés pour effectuer des calculs sur 8 bits pour les couches de base utilisées dans les réseaux neuronaux. Les TPU sont optimisés pour certaines couches, Le

développement rapide des réseaux neuronaux pourrait potentiellement rendre la conception obsolète. En outre, la conception et la fabrication des TPU sont coûteuses par rapport aux autres alternatives.

	Architecture	**Parallelism**	**Memory**	**TOP/s/W**	**Précision**
CPU	basic	low	12MB	x1	Float
GPU	Co-processor	high	12GB	x1.2-4	Float
FPGA	streaming/dedicated/co-processor	Medium	512MB-4GB	x90-244	1-16 bit
TPU	Co-processor	High	16GB	x127	8 bit

Tableau 1. Comparaison d'exécution matérielle d'un CNN sur différentes plateformes

7. Apprentissage d'un réseau de neurones:

Le processus d'apprentissage d'un réseau de neurones consiste à ajuster, optimiser et adapter les paramètres θ (les poids W et les termes biais b) de chaque couche en minimisant l'erreur de prédiction jusqu'à atteindre l'état désiré du réseau [7]. Les réseaux de neurones sont souvent appris à l'aide d'un algorithme de descente du gradient. Trois types de variantes de gradient ont été proposés dans la littérature :

7.1. Descente de gradient classique :

Le gradient est au niveau de l'ensemble du corpus d'apprentissage de N instances. Le gradient est calculé au niveau du corpus d'apprentissage (de N instances) complet qui rend le processus d'apprentissage coûteux en termes de ressources et de temps. Les paramètres θ sont mis à jour comme suit :

$$\theta_{t+1} = \theta t - \eta \frac{1}{N} \sum_{i=1}^{N} \frac{\partial C(\hat{y}^{i}, y^{i})}{\partial \theta}$$

Où η le taux d'apprentissage (learning rate) ; y (références) ; yˆ (hypothèses) ; C (coût).

7.2. Descente de gradient stochastique (SGD) :

Le gradient est calculé sur chaque instance du corpus d'entrainement. Cela rend le processus d'apprentissage coûteux en termes de temps et de ressources. De plus, si le corpus d'entrainement est bruité, le SDG peut être instable et renvoie ainsi de mauvaises prédictions. Les nouveaux paramètres θ sont obtenus comme suit :

$$\boldsymbol{\theta t + 1 = \theta t - \eta + \frac{\partial C(\hat{y}^{i}, y^{i})}{\partial \theta}}$$

7.3. Descente de gradient mini-batch :

C'est une variante de l'algorithme SGD. Le gradient est calculé sur un ensemble de m instances (mini-batch) du corpus d'entrainement au lieu d'une seule instance. C'est l'algorithme le plus utilisé aujourd'hui, il est beaucoup plus efficace et rapide par rapport aux autres algorithmes. La mise à jour des paramètres s'effectue comme suit :

$$\boldsymbol{\theta^{i+\&} = \theta^{i} - \eta \frac{1}{m} \sum_{i=1}^{m} \frac{\partial C(\hat{y}^{i}, y^{i})}{\partial \theta}}$$

Une fois que la fonction de coût et la méthode de descente de gradient sont définies, le processus d'apprentissage est ensuite effectué à l'aide de l'algorithme rétro-propagation du gradient. C'est un algorithme récursif de calcul du gradient, où l'apprentissage s'effectue en deux étapes :

1. Une *propagation avant* qui consiste à parcourir le réseau de la première couche vers la dernière couche L en calculant les sorties d'activation f_θ^l de chaque couche $l \in [1,...,L]$. Sachant que $f_\theta{}^l = W^l \cdot f_\theta{}^{l-1} + b^l$ lorsque $l > 1$;

2. Une *propagation arrière* : la propagation arrière est effectuée de la dernière couche vers la première couche après avoir calculé la fonction de coût $C(y,\hat{y})$. Ainsi, la première étape de propagation consiste à calculer le gradient $\frac{\partial C}{\partial f_\theta^L}$ de la fonction de coût par rapport à la sortie de la dernière couche L du réseau.

Ensuite, le gradient $\frac{\partial C}{\partial \theta^l}$ de la fonction *C* par rapport aux paramètres θ^l est calculé pour chaque couche *l* comme suit : $\frac{\partial C}{\partial \theta^l} = \frac{\partial f_\theta^l}{\partial \theta^l} \frac{\partial C}{\partial f_\theta^l}$

Conclusion :

Dans ce chapitre, nous avons abordé le réseau de neurones qui est l'élément de base du CNN, ainsi que les fonctions d'activation. Nous avons détaillé les différentes couches du CNN de façon générale. Nous avons également défini les deux parties software Framework et hardware nécessaires pour la création d'un réseau de neurones. Le chapitre suivant abordera les éléments de base des architectures matérielles des CNN sur FPGA

Chapitre 3 : CNN et FPGA

Chapitre 3 : CNN et FPGA

1. Introduction :

La popularité des CNN et des implémentations potentielles sur les dispositifs embarqués a motivé les chercheurs à étudier les possibilités de mettre en œuvre des CNN sur les FPGA. Les FPGA peuvent offrir des performances en temps réel, une efficacité énergétique élevée et conception flexibles. De nombreuses études ont mis en œuvre différentes approches pour améliorer l'optimisation des performances et la consommation d'énergie.

2. Les FPGAs :

FPGA c'est un circuit logique programmable, ou réseau logique programmable, est un circuit intégré logique qui peut être reprogrammé après sa fabrication [8].

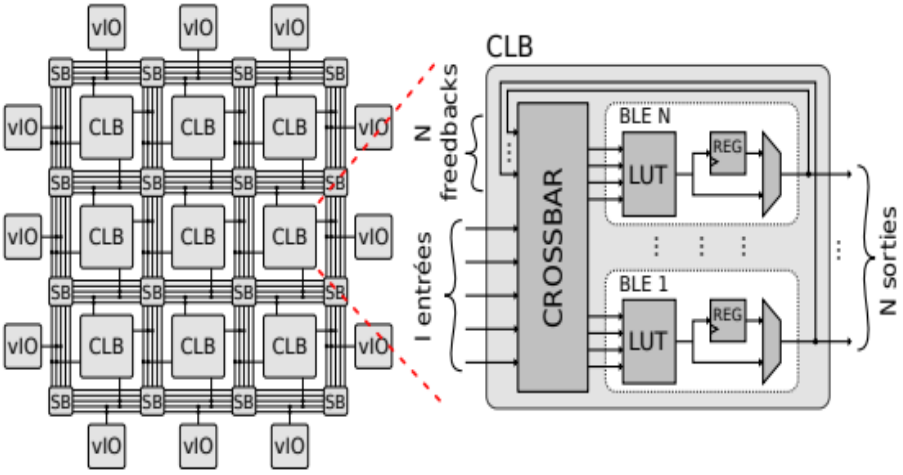


Figure 3.1 : Exemple de composition d'un FPGA.

2.1. Les blocs logiques configurables CLB :

Les blocs logiques configurables sont les éléments déterminants des performances du FPGA. Chaque bloc est composé d'un bloc de logique combinatoire composé de deux générateurs de fonctions à quatre entrées et d'un bloc de mémorisation synchronisation composé de deux bascules D. Quatre autres entrées permettent d'effectuer les connexions internes entre les différents éléments du CLB. La figure ci-dessous, nous montre le schéma d'un CLB.

2.2. LUT (Look Up Table) :

Une LUT, qui signifie Look Up Table, est en général une table qui détermine la sortie pour une ou plusieurs entrées données. Dans le contexte de la logique combinatoire, il s'agit de la table de vérité. Cette table de vérité définit effectivement le comportement de votre logique combinatoire.

2.3. Blocs d'E/S configurables :

Les IOB (input output bloc): Ces blocs entrée/sortie permettent l'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisé.

2.4. Les différents types d'interconnexions :

Les connexions internes dans les circuits FPGA sont composées de segments métallisés. Parallèlement à ces lignes, nous trouvons des matrices programmables réparties sur la totalité du circuit, horizontalement et verticalement entre les divers CLB. Elles permettent les connexions entre les diverses lignes, celles-ci sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM. Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé possible.

2.5. Circuit d'horloge :

Des blocs d'E / S spéciaux avec des tampons d'horloge à entraînement élevé spéciaux, appelés pilotes d'horloge, sont répartis autour de la puce. Ces tampons se connectent aux pads d'entrée d'horloge et entraînent les signaux d'horloge sur les lignes d'horloge globales. Ces lignes d'horloge sont conçues pour des temps d'inclinaison faibles et des temps de propagation rapides. Notez que la conception synchrone est une obligation avec les FPGA, car le décalage absolu et le retard ne peuvent être garantis nulle part ailleurs que sur les lignes d'horloge.

2.6. Cœurs de processeur :

Les cœurs de processeur sont l'un des types de cœurs couramment disponibles en tant que cœurs IP(IntellectualProperty) ou noyaux intégrés. Ces processeurs sont généralement ceux qui sont conçus pour les systèmes embarqués car, presque par définition, les appareils programmables sont des systèmes embarqués.

2.7. Cœurs DSP :

Les processeurs de signaux numériques (DSP) sont un autre type commun de cœur qui est proposé sous forme de cœur IP ou de cœur intégré. Ce sont essentiellement des processeurs spécialisés qui sont utilisés pour manipuler des signaux analogiques. Ils sont couramment utilisés pour le filtrage et la compression des signaux vidéo ou audio. Suivant le travail réalisé, il est important de définir le type de DSP utilisé lors de la conception du réseau CNN à base d'FPGA globales.

3. Implémentation d'un réseau CNN sur FPGA en utilisant la synthèse de haut niveau (HLS) : aperçu générale :

Au cours des 30 dernières années, les circuits électroniques mis en œuvre dans les FPGA ont été conçus à l'aide de langages de description de matériel (HDL) dans lesquels le comportement de chaque pièce de matériel est décrit en détail, ce qui prend un temps considérable pour le développement tout en ayant des informations très détaillées sur le matériel conçu [8]. Ces dernières années, une autre approche de la conception de matériel gagne en popularité : la description est faite dans un langage de programmation de haut niveau comme le C et il existe un outil chargé de générer une description dans un langage RTL, en extrayant les informations du langage de haut niveau. Cette nouvelle approche est connue sous le nom de synthèse de haut niveau (HLS).

La HLS n'a pas été populaire dans le passé en raison des mauvais résultats qu'elle obtenait et de la crainte des concepteurs de perdre le contrôle du matériel généré. Cependant, aujourd'hui, la plupart des outils qui existent sur le marché à cette fin ont atteint un niveau

de maturité et de stabilité qui permet une compréhension suffisante du matériel généré, tout en étant généralement plus efficace que le code des développeurs, ou du moins suffisamment pour compenser le coût d'un cycle de développement plus court par rapport au HDL. Pour toutes ces raisons, HLS a été la technologie choisie pour ce mémoire.

La question la plus souvent posée au sujet de la HLS, en particulier lorsque le lecteur a une certaine connaissance de l'idéal d'un langage HDL et des différences entre le matériel et le logiciel, est probablement de savoir comment il est possible de modéliser le matériel en utilisant un langage orienté logiciel. En d'autres termes, comment est-il possible de passer des boucles for et while, des instructions if, des fonctions, des appels et même de certains principes orientés objet à des machines d'état, des compteurs, des mémoires et des signaux ? Ce n'est évidemment pas simple, mais cela nécessite une réécriture complète du code pour permettre la parallélisation matérielle. Cependant, cela n'est généralement pas suffisant pour générer des circuits personnalisés. Pour cela, tous les outils incluent un certain nombre de pragmas de compilation qui contrôlent le déroulement des boucles, la création de pipelines, l'organisation de la mémoire, etc. qui donnent au concepteur un meilleur contrôle sur la façon dont le matériel sera mis en œuvre.

Dans le cadre de ce mémoire, l'objectif est de mettre en œuvre un algorithme d'apprentissage profond pour la reconnaissance des chiffres manuscrits en tirant parti de la puissance du calcul parallèle à l'intérieur d'une structure FPGA, l'accélération du processus de développement se fait à l'aide de la synthèse de haut niveau (HLS), pour cela, le schéma suivi est montré sur la figure suivante :

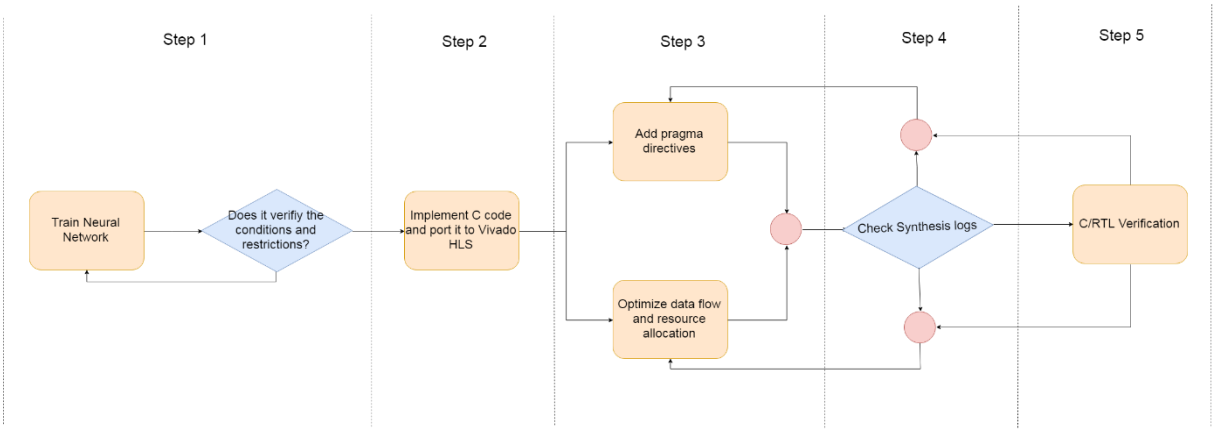


Figure 3.2 : Présentation du schéma de travail

Chapitre 3 : CNN et FPGA

3.1. Le choix d'une architecture à réseaux de neurones :

Nous avons d'abord exploré quelques architectures de réseaux neuronaux différentes et choisi celle qui était la mieux adaptée à la tâche à accomplir en satisfaisant un ensemble de critères de performance et de restrictions de conception. Nous avons utilisé Python pour l'entrainement du réseau neuronal car il existe de nombreuses ressources disponibles en ligne. L'entrainement nécessite beaucoup de temps et un autre avantage de l'utilisation de Python est l'accès à un certain nombre de services de cloud computing pour accélérer l'entrainement, comme FloydHub, Amazon Web Services et l'Intel AI Academy.

Chaque réseau neuronal possède un ensemble de paramètres (poids), qui sont formés, et des hyperparamètres, qui sont choisis de manière plus ou moins empirique. Certains de ces hyperparamètres ont une influence directe sur la structure du réseau neuronal :

- Le nombre de couches convolutionnelles ;
- Nombre de couches entièrement connectées (FC) ;
- Nombre d'unités (neurones) dans chaque couche FC ;
- Taille du noyau, stride, padding ;
- Nombre de noyaux (kernels) ;
- Fonction d'activation.

Les autres hyperparamètres n'affectent que le processus de formation :

- Dropout ;
- L'entrainement du mini-batch ;
- taille du mini-batch ;
- Taux d'apprentissage ;
- Algorithme de minimisation.

3.2. Implémentation du réseau de neurone en C :

Pour pouvoir déployer l'algorithme du réseau neuronal sur un FPGA, l'algorithme doit être écrit dans un langage de description du matériel. C'est pourquoi il faut réécrire

manuellement toute l'étape d'inférence du réseau neuronal en C/C++. Pour s'assurer que le code C nouvellement implémenté fonctionnait correctement, il a été testé par rapport à l'inférence Python en utilisant les mêmes poids entrainé.

3.3. Ajout des éléments HLS :

HLS est livré avec un certain nombre de directives #pragma qui permettent à l'utilisateur de guider le compilateur HLS sur la façon de générer la conception souhaitée. Il contient également un certain nombre de bibliothèques contenant des types de données et des fonctions prédéfinis qui peuvent améliorer les performances de la conception.

3.4. Synthèse et validation :

Dans HLS, le terme synthèse signifie la conversion du code C en Verilog ou VHDL. À la fin d'un processus de synthèse réussi, on obtient des dossiers RTL contenant du code Verilog et VHDL et un journal de synthèse contenant des informations sur la consommation, la latence et la fréquence d'horloge. La conception doit être ajustée jusqu'à ce que les performances matérielles souhaitées soient atteintes. Enfin le code C devient plus optimise par rapport auquel le RTL généré est testé. Un testbench en C est utilisé pour générer et vérifier les stimuli.

Conclusion :

Dans ce chapitre, nous avons abordé tous les éléments nécessaires afin de réaliser l'implémentation d'un réseau de neurones convolutif sur FPGA, le réseau de neurones choisi a pour objectif de reconnaitre des chiffres manuscrits en utilisant une synthèse HLS. Dans le chapitre suivant nous montrons les étapes de la réalisation du projet.

Chapitre 4 : Accélération matérielle et implémentation CNN sur FPGA

1. Introduction :

Les réseaux de neurones à convolution profonde (CNN) sont les systèmes de pointe pour la classification d'images en raison de leur grande précision. L'accélération est aujourd'hui la cible dans ce domaine pour utiliser ces systèmes dans des applications temps réel. Les unités de traitement graphique sont la solution, mais sa consommation d'énergie élevée empêche son utilisation dans les équipements utilisés quotidiennement. De plus, le FPGA a une faible consommation d'énergie et une architecture flexible qui convient davantage aux implémentations CNN. Dans ce travail, nous implémentons sur une carte FPGA de type PYNQ Z2 (Xilinx), un model CNN prédéfini qui consiste en une calculatrice mathématique manuscrite, ce modèle a pour entrer une image qui contient des opérations mathématiques entre des chiffres écrits à la main, cette image doit contenir 5 parties : deux opérations et 3 chiffres, commençant d'abord par un chiffre, une opération, un deuxième chiffre, une deuxième opération et un troisième chiffre (figure 4.1). Le CNN reconnait le premier chiffre (de 0 à 9), la première opération (+,-,* ou /), le deuxième chiffre, la deuxième opération et le troisième chiffre de la même manière. Finalement, le model calcule le résultat des opérations et l'affiche. La mise en œuvre de ce travail d'implémentation a été initié par le logiciel ISE Xilinx, où nous avons utilisé vivado HLS (High LevelSynthesis) pour le synthèse et l'estimation des ressources occupées par l'FPGA, puis nous avons généré le bitstream que nous utilisons pour l'implémentation. Ce chapitre détaille les étapes suivies afin de réaliser cette dernière.

2. Le Modèle CNN utilisé :

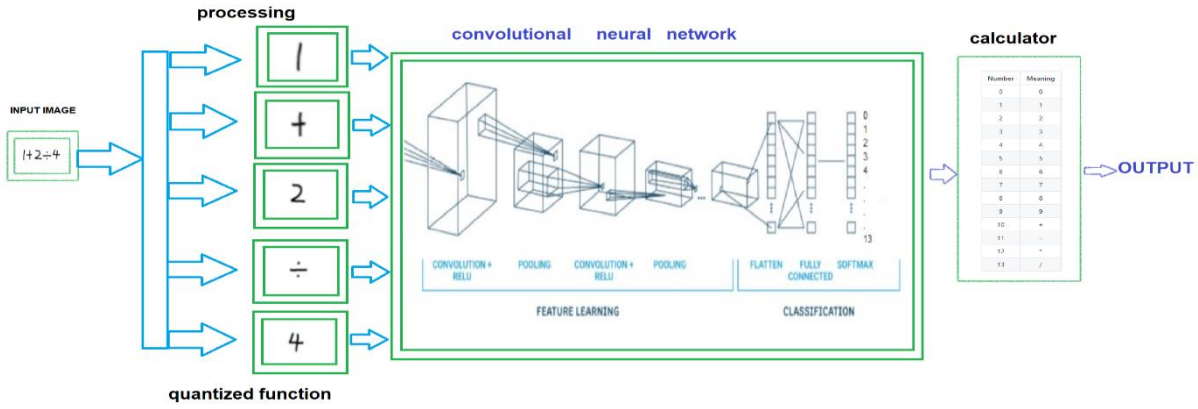


Figure 4 .1 : Le modèle CNN implémenté

Dans ce projet nous avons utilisé le modèle CNN prédéfini présenté dans la figure 4.1, Ce modèle est composé de 2 couches de covolution + RELU et 2 couches de max pooling, une couche pour le fully connected et une classification de type softmax. Les paramètres du CNN sont détaillés dans le tableau suivant.

1ère couche de Convolution	Filter weight(5x5,1,32)	bias (32)	strides=[1,1,1,1]	padding= SAME + Relu
Max Pooling	ksize=[1,2,2,1]		strides=[1,2, 2,1]	padding=SAME
2ème couche de Convolution	Filter Weight(5,5,32, 64)	bias (64)	strides=[1,1,1,1]	padding= SAME + Relu
Max Pooling	ksize=[1, 2, 2, 1]		strides=[1,2,2,1]	padding= SAME
Flatten layer	weight(8*8*64, 1024)	bias (1024)		
Fully connected	weight(1024, 14)	bias (14)		

Tableau 2 : paramètres du modèle CNN implémenté

3. La présentation de la carte PYNQ Z2

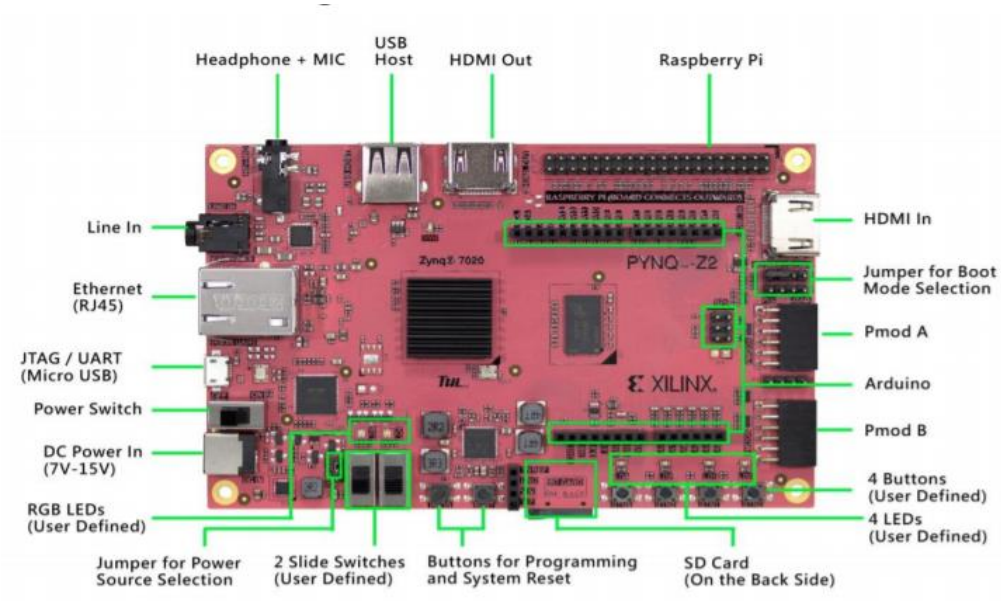


Figure 4.2 : l'emplacement des composants de la carte PYNQ Z2

PYNQ-Z2 est une carte de développement FPGA basé sur ZYNQ XC7Z020 FPGA, intensément conçu pour soutenir PYNQ, un projet open-source de Xilinx® qui facilite la conception de systèmes embarqués avec les SoC (Systems on Chips) Xilinx Zynq. Ses caractéristiques sont définies sur la figure 4.3.

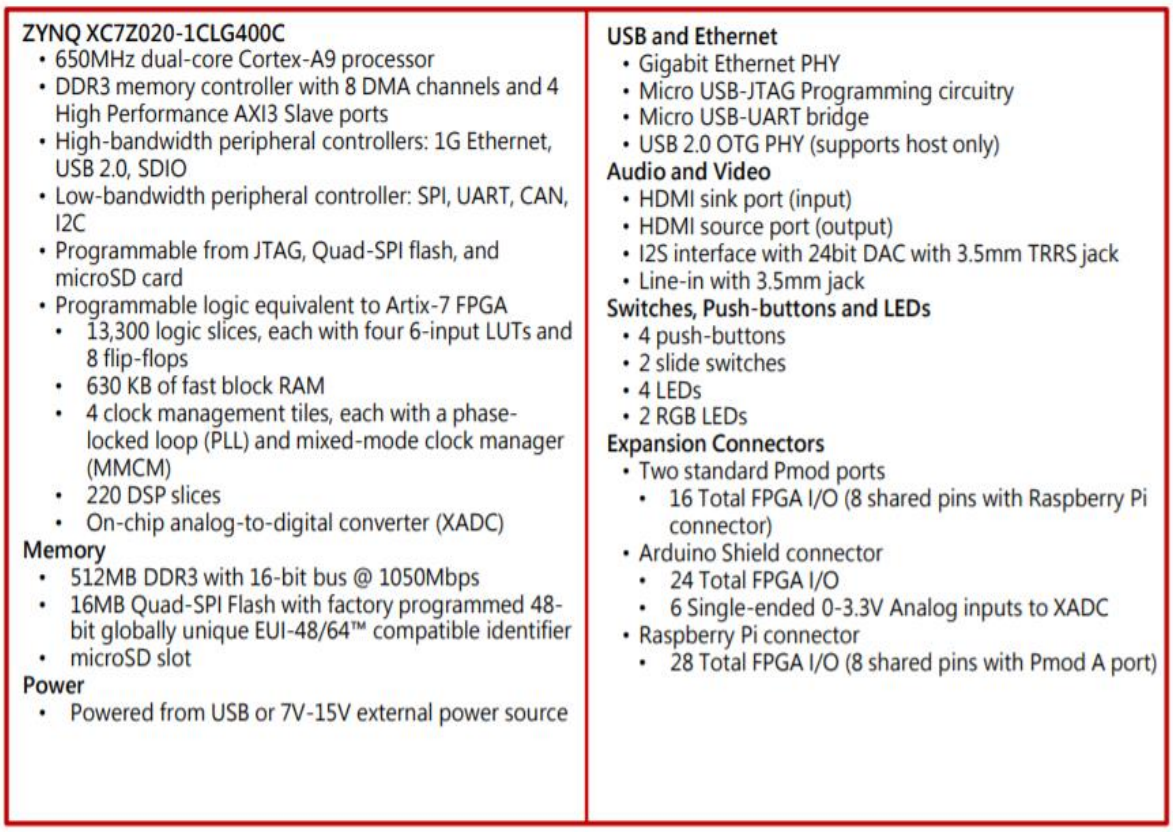


Figure 4 .3: les composants de la carte PYNQ Z2

4. Les overlays :

Le concept d'overlay est relativement récent, et il n'a pas encore de définition précise et arrêtée. La définition la plus générique d'un overlay que l'on puisse donner est la suivante : Un overlay est un design reconfigurable implémenté sur FPGA (qui est lui-même reconfigurable). En d'autres termes, un overlay est une couche d'abstraction matérielle placée entre l'application le FPGA hôte, isolant celle-ci de ce dernier. [9]

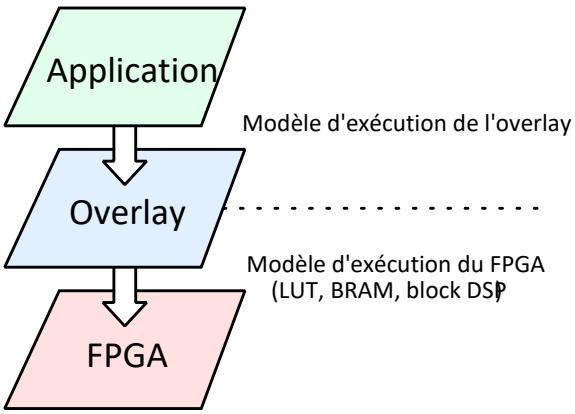


Figure 4 .4: L'overlay isole l'application du FPGA sous-jacent. Il apporte son propre modèle d'exécution.

Comme illustré figure 4.4, l'overlay apporte son propre modèle d'exécution, qui peut être radicalement différent de celui du FPGA sous-jacent. L'application ne voit que le modèle d'exécution de l'overlay et est isolée du FPGA hôte. Le modèle d'exécution de l'overlay détermine sa granularité. Les overlays permettent d'explorer l'espace entre la facilité d'usage des processeurs et les performances des FPGAs. Les overlays sont donc de bons candidats pour être utilisés en tant qu'accélérateurs de calcul reconfigurables

4.1. Intégration des overlays :

La gestion bas niveau de l'overlay est réalisée aux niveaux matériel et logiciel : un ensemble de contrôleurs matériels vient s'interfacer avec l'overlay pour permettre sa gestion par un logiciel que nous appelons hyperviseur. L'overlay et ses contrôleurs matériels sont rassemblés en une IP. L'hyperviseur étant logiciel, un processeur est nécessaire pour l'exécuter.

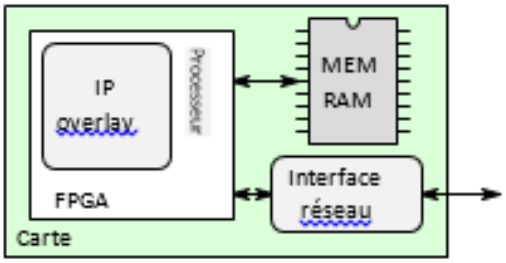


Figure 4 .5: Exemple d'intégration de l'overlay, l'hyperviseur est exécuté sur le processeur dans le FPGA

Dans le cas de la carte Pynq Z2, le FPGA est utilisé seul c'est à dire en mode "standalone", c'est-à-dire qu'il est utilisé seul sans processeur et qu'il est directement relié au réseau. Dans ce cas, l'hyperviseur doit être exécuté sur le FPGA sur un processeur softcore implémenté via les ressources reconfigurables du FPGA. Ce cas est illustré figure 4.5.

4.2. Intégration de l'IP overlay avec processeur physique

Lorsqu'un processeur physique est présent dans la plateforme, il peut être soit externe au FPGA, soit embarqué de manière fixe dans le FPGA avec un ensemble de périphériques. Si le processeur est embarqué dans le FPGA(cas de la carte Pynq Z2), comme illustré figure 4.6, alors il a un accès direct à la matrice reconfigurable du FPGA, et l'intégration de l'IP overlay ne demande qu'un adaptateur entre les interfaces Wishbone maître et esclave de l'IP et les ports de connexions à la partie fixe, ainsi que de connecter le signal d'interruption généré par l'IP au contrôleur d'interruption du processeur dans la partie fixe. Par exemple, dans le cas de la famille de FPGA Zynq de Xilinx d'où la carte PYNQ Z2, l'intégration de l'IP demande un adaptateur Wishbone → AXI et un adaptateur AXI → Wishbone. Le processeur peut ainsi avoir accès aux registres de configuration de l'IP, et l'IP et le processeur partagent l'accès à une même mémoire RAM

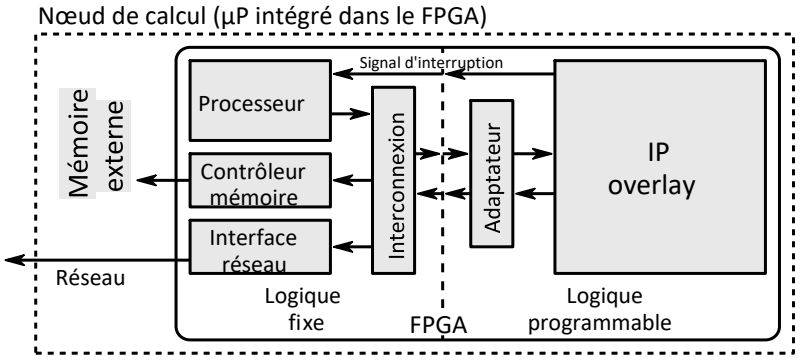


Figure 4 .6: Intégration de l'overlay dans un FPGA comportant un processeur implémenté sur le silicium : cas de la carte PYNQ Z2.

4.3. Overlay de base

Le but de la conception de l'overlay de base est de permettre à PYNQ d'utiliser des périphériques sur une carte prête à l'emploi. La conception inclut une adresse IP matérielle pour

contrôler les périphériques sur la carte cible et connecte ces blocs IP au Zynq PS. Si l'overlay de base est disponible pour une carte, les périphériques peuvent être utilisés à partir de l'environnement Python immédiatement après le démarrage du système.

Les périphériques de la carte incluent généralement des périphériques GPIO (voyants, commutateurs, boutons), vidéo, audio et autres interfaces personnalisées.

Comme l'overlay de base inclut IP pour les périphériques sur une carte, elle peut également être utilisée comme une référence pour créer de nouvelles overlay personnalisées.

Dans le cas des interfaces à usage général, par exemple les interfaces Pmod ou Arduino, l'overlay de base peut inclure un PYNQ MicroBlaze. Un PYNQ MicroBlaze permet de contrôler des appareils avec différentes interfaces et protocoles sur le même port sans nécessiter de modification de la conception de la logique programmable.

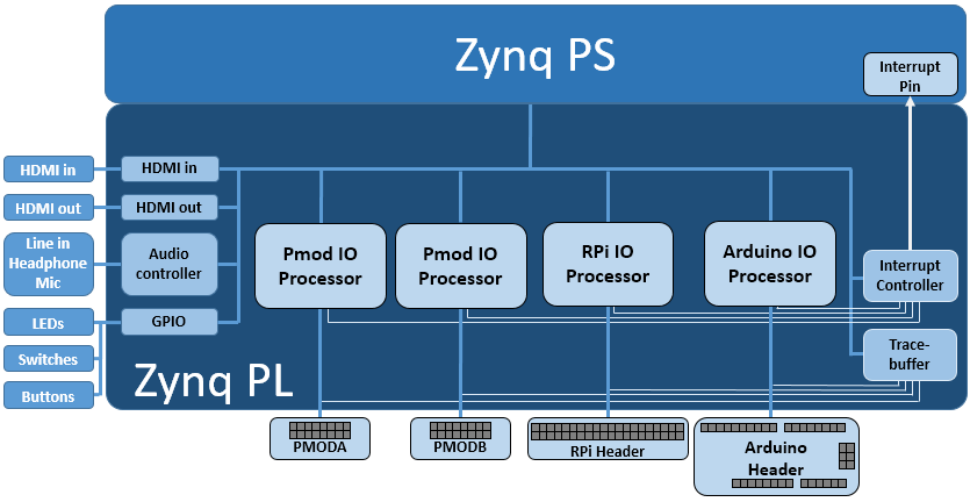


Figure 4.7 : Overlay de base

4.4. Overlay logictools

L'overlay logictools se compose de blocs matériels programmables à connecter à des circuits logiques numériques externes. Des machines à états finis, des fonctions logiques booléennes et des modèles numériques peuvent être générés à partir de Python. Un commutateur programmable connecte les entrées et les sorties des blocs matériels aux broches d'E / S externes. L'overlay logictools peut également avoir un analyseur de trace pour capturer les données de l'interface IO pour l'analyse et le débogage.

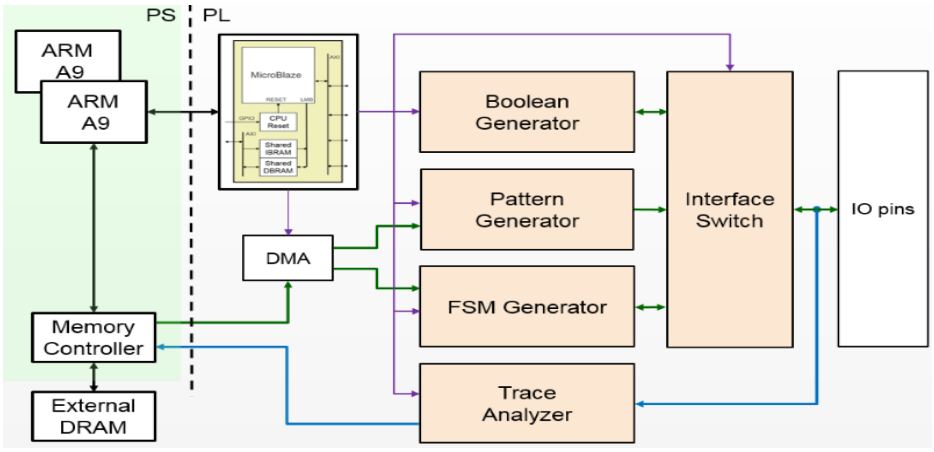


Figure 4.8 : Overlay logictools

5. Les logiciels utilisés :

5.1. Microsoft visual studio :

Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications web ASP.NET, des services web XML, des applications bureautiques et des applications mobiles. Visual Basic, Visual C++, Visual C# utilisent tous le même environnement de développement intégré (IDE), qui leur permet de partager des outils et facilite la création de solutions faisant appel à plusieurs langages. Par ailleurs, ces langages permettent de mieux tirer parti des fonctionnalités du framework .NET, qui fournit un accès à des technologies clés simplifiant le développement d'applications web ASP et de services web XML grâce à Visual Web Developer.

5.2. Vivado :

Vivado Design Suite est une suite logicielle produite par Xilinx pour la synthèse et l'analyse des conceptions HDL , remplaçant Xilinx ISE avec des fonctionnalités supplémentaires pour le développement de système sur puce et la synthèse de haut niveau . Vivado représente une réécriture et une refonte de tout le flux de conception.

5.3. Win32 Disk Imager :

Win32 Disk Imager est un utilitaire permettant de transférer l'**image** disque d'un système d'exploitation sur une carte mémoire SD. Idéal dans le cas où vous cherchez à créer une carte mémoire d'installation pour votre Raspberry Pi, **Win32 Disk Imager** est extrêmement simple à utiliser.[18]

5.4. jupyter :

Jupyter est une application web utilisée pour programmer dans plus de 40 langages de programmation, dont Python, Julia, Ruby, R, ou encore Scala2. C'est un projet communautaire dont l'objectif est de développer des logiciels libres, des formats ouverts et des services pour l'informatique interactive.

5.5. TENSORFLOW:

TensorFlow est une plate-forme open source de bout en bout pour l'apprentissage automatique. Il dispose d'un écosystème complet et flexible d'outils, de bibliothèques et de ressources communautaires qui permet aux chercheurs de pousser l'état de l'art en matière de ML et aux développeurs de créer et de déployer facilement des applications basées sur le ML.

TensorFlow a été initialement développé par des chercheurs et des ingénieurs travaillant au sein de l'équipe Google Brain au sein de l'organisation Machine Intelligence Research de Google pour mener des recherches sur l'apprentissage automatique et les réseaux de neurones profonds. Le système est suffisamment général pour être également applicable dans une grande variété d'autres domaines. TensorFlow fournit des API Python et C++ stables, ainsi qu'une API rétro compatible non garantie pour d'autres langages. [10]

6. Les étapes de la réalisation du projet :

La réalisation de ce projet est faite selon trois étapes principales :

6.1. La première étape est la génération du CORE IP :

Premièrement on va ouvrir le logiciel VIVADO HLS.

Après on va créer un nouveau projet.

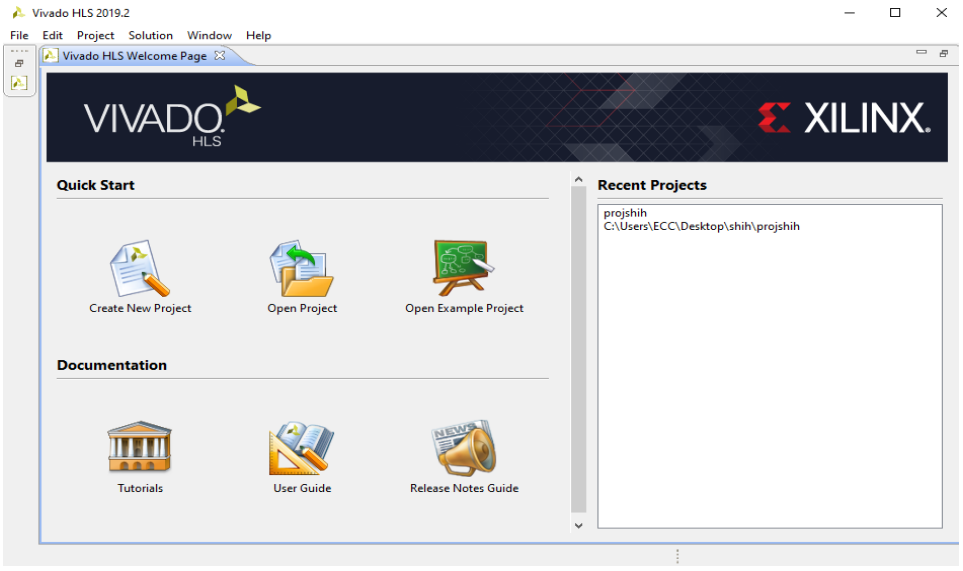


Figure4.9 : un nouveau projet HLS

Ensuite on choisit la carte désirée qui est la carte PYNQZ2 pour notre projet

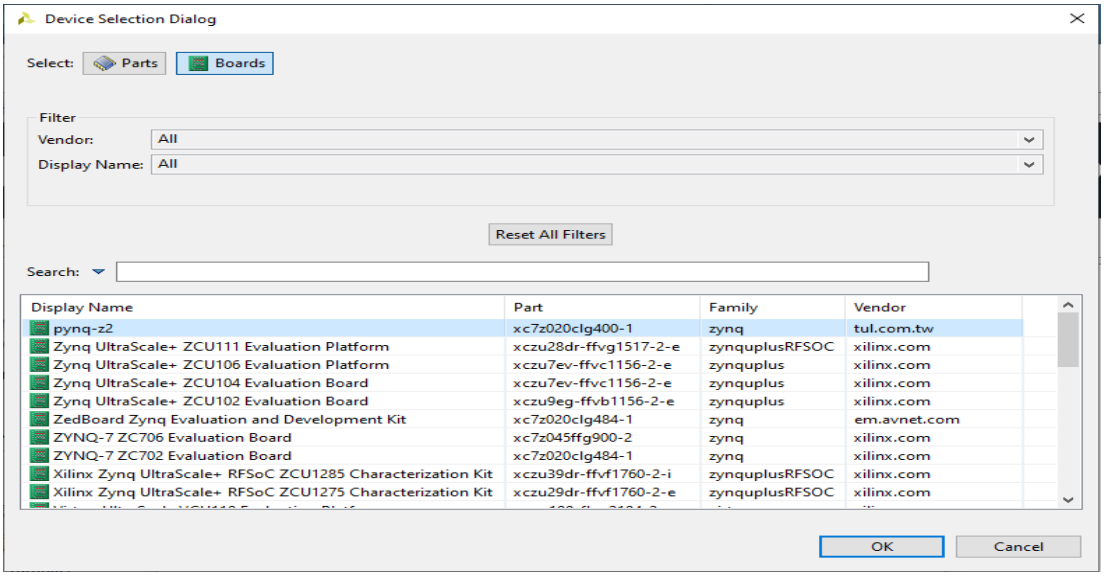


Figure 4.10 : Le choix de la carte PYNQZ2 dans HLS

Après la création du projet on ajoute les fichiers ‘’cnn.h’’ et ‘’cnn.cpp’’ comme des fichiers source et le fichier ‘’main.cpp’’ et les deux dossiers ‘’input_imgs’’ et ‘’parameters’’ dans la catégorie testbench. Les fichiers cnn.h et cnn.cpp sont les codes sources du CNN prédéfini qui a été choisi pour implémentation.

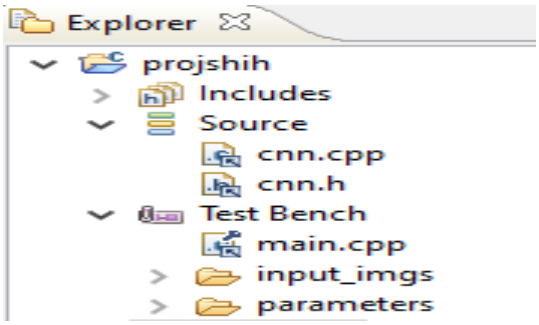


Figure 4.11 : les fichiers ajoutés

Ensuite on va sélectionner la fonction ‘’ detection_acc ’’ comme fonction du top et démarrer la synthèse pour obtenir le code RTL.

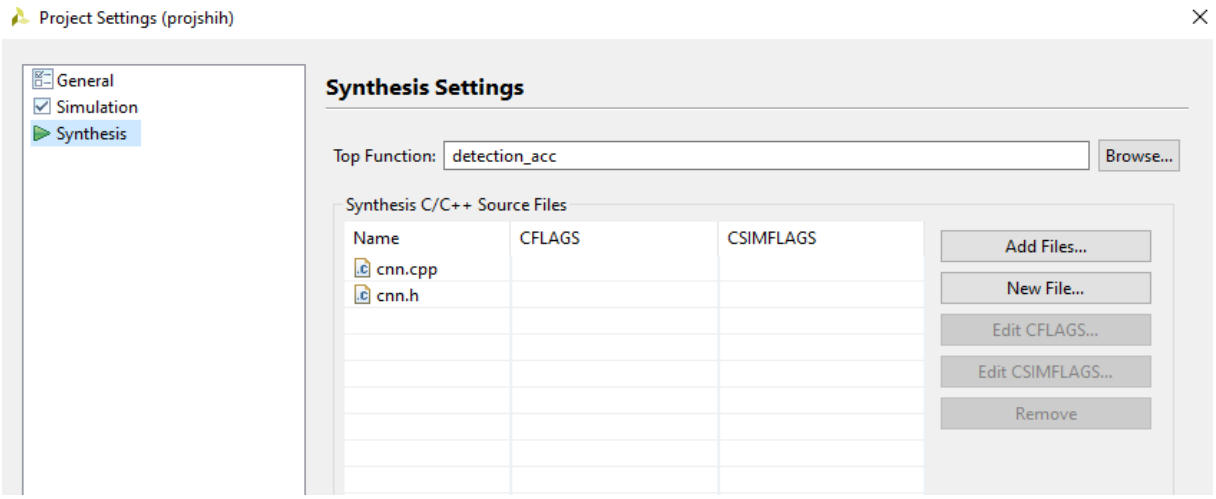


Figure 4.12 : Top function

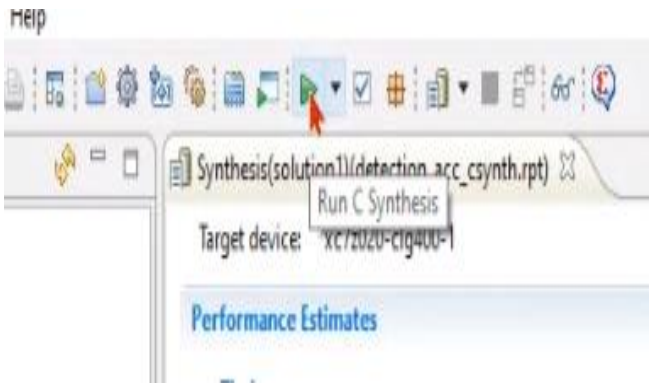


Figure 4.13: RUN C Synthesis

Enfin on va cliquer sur export RTL pour générer le CORE IP depuis le code RTL obtenu.

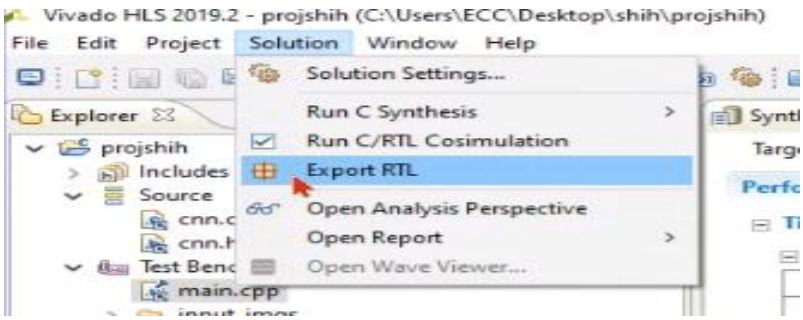


Figure 4.14 : Export RTL

6.2. La 2eme étape est la création du BLOC DESIGN :

Premièrement la Création du nouveau projet vivado

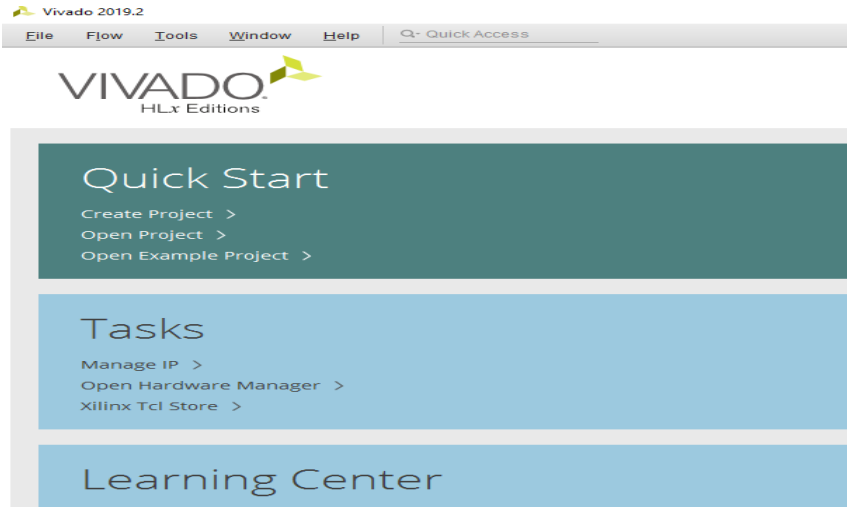


Figure 4.15 : un nouveau projet vivado

On sélectionne la carte désirée (PYNQ-Z2)

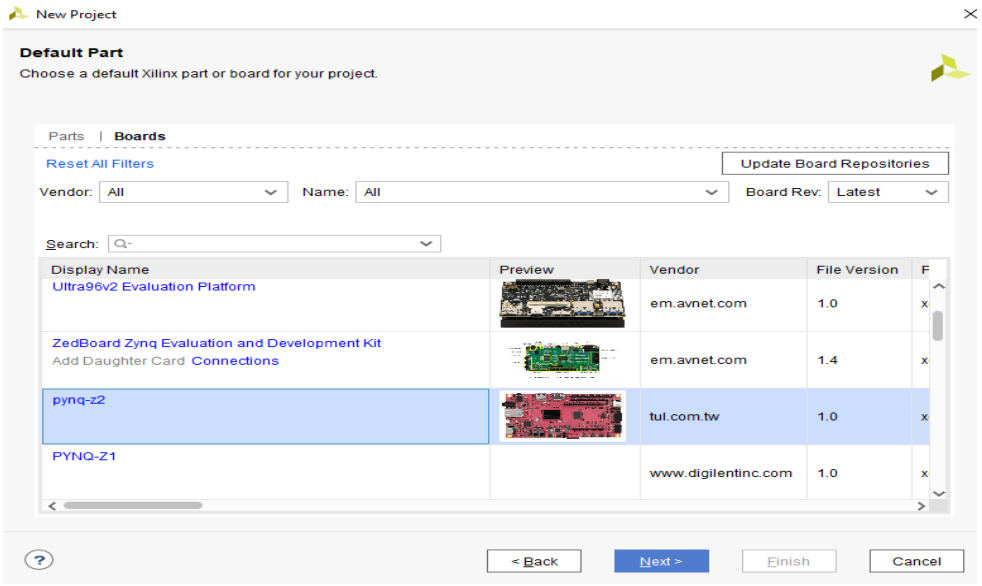


Figure 4.16 : choisir la carte PYNQZ2 dans vivado

Ensuite on va importer le dossier IP du HLS au dossier du IP actuelle.

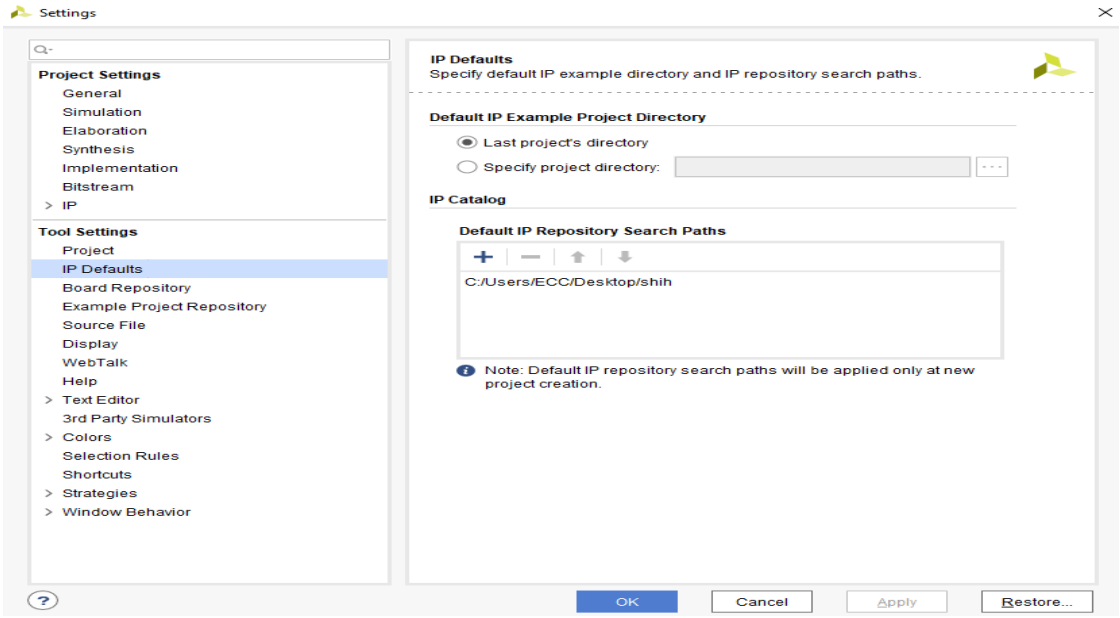


Figure 4.17 : importe IP du HLS

Après on réalise le BLOC DESIGN en faisant les connexions nécessaires et on le valide

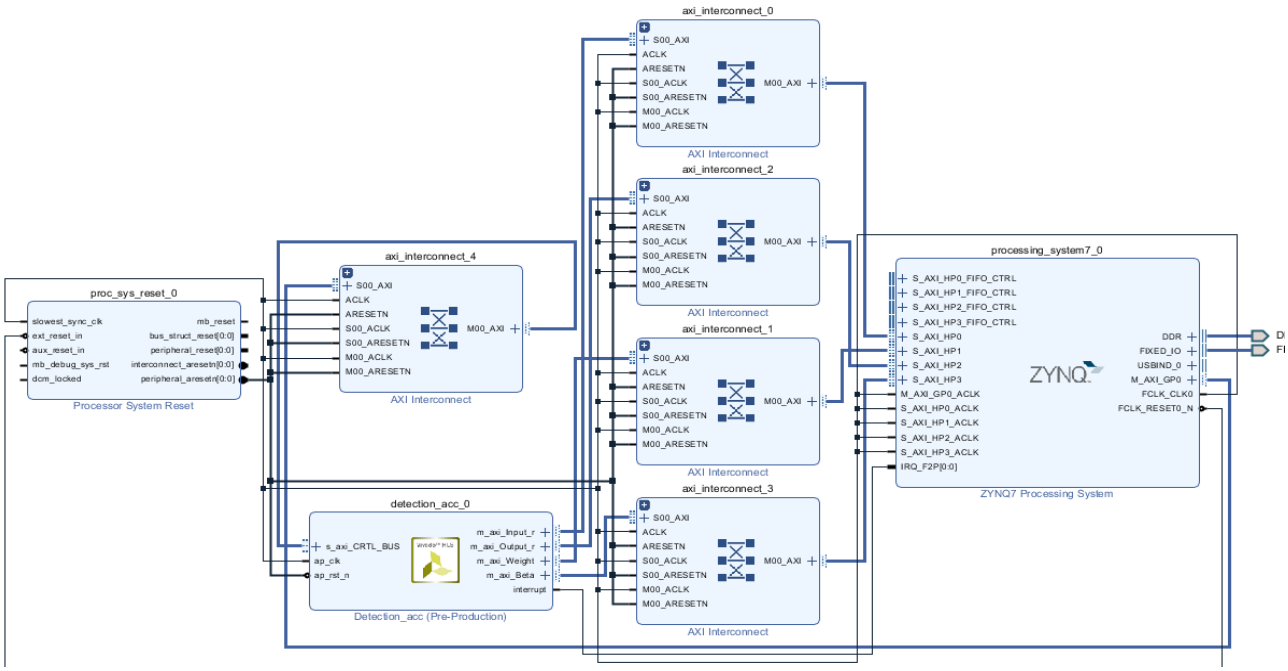


Figure 4.18 : BLOC DESIGN

Enfin on va exporter le bloc design et générer le bitstream pour obtenir les fichiers ‘’.bit’’ (bitstream) et ‘’.hwh’’ (hwh fait référence à Hardware handoff).

Les deux fichiers .bit et .hwh sont générés lors de la génération du bitstream. Le fichier hwh contient les informations du Block Design, il permet d'extraire toutes les informations nécessaires à la création d'une application ciblée. Le fichier bit contient des informations binaires qui se transmettent à l'FPGA pour le programmer.

- Le bloc IP « overlay » qu'on a créé nommé « detection_acc » (figure 4.18) est le bloc qui définit le modèle CNN et qui va effectuer son accélération.
- Les AXI interconnect (Advanced eXtensible Interface) sont des protocoles de communication entre les bloc IP d'un FPGA, ils ont l'avantage d'avoir des bus d'écriture et de lecture séparés, ce qui entraîne l'accès direct à la mémoire (DMA) à faible coût, en plus la possibilité d'avoir des transactions qui peuvent être effectuées dans le désordre.

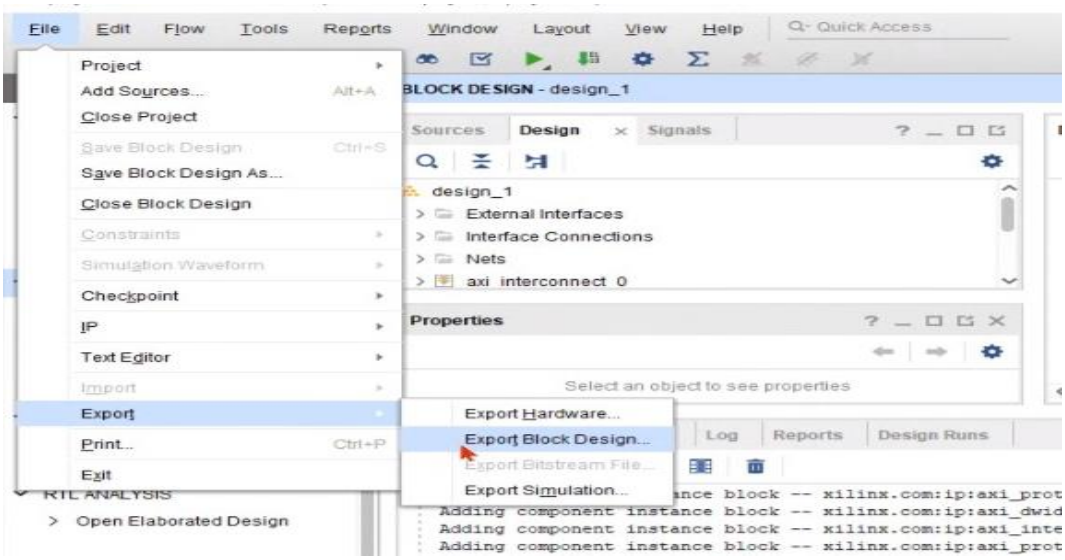


Figure 4.19 : Export du bloc design

6.3. La 3eme étape L'exécution sur PYNQ-Z2:

Les Préparations pour PYNQ z2 :

Nous gravons l'image v2.5 (téléchargeable sur internet) spécialement conçue pour PYNQ-Z2 sur sa carte SD, cette image contient tous les packages et informations nécessaires afin de faire fonctionner la carte FPGA PYNQ Z2. Pour se faire, nous utilisons le logiciel Win32 Disk lmager.

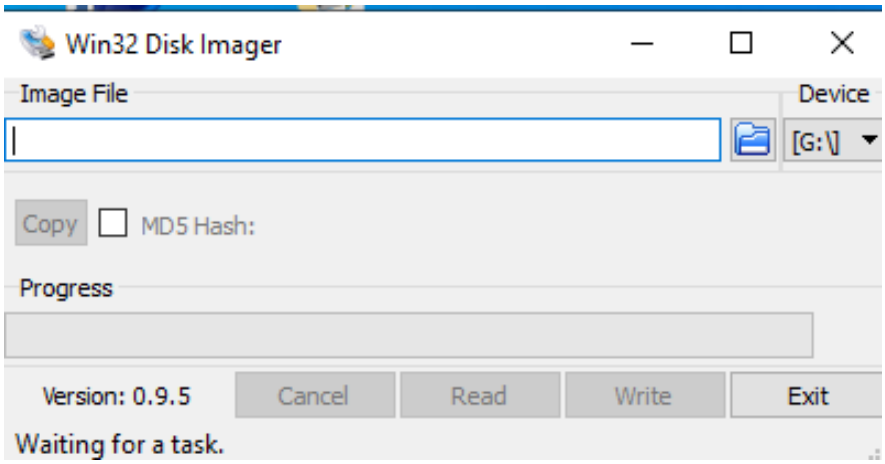


Figure 4.20 : Win 32 Disk lmager

Une fois fait, nous allons par la suite copier dans la même carte SD, les fichiers de test (images de test) afin de tester notre modèle ainsi que les fichiers d'exécution à savoir le fichier bitstream et le fichier HWH générés par Vivado et HLS, ainsi que le fichier qu'on a appelé HMC.ipynb afin de pouvoir exécuter les étapes pour l'implémentation.

Ensuite, nous connectons le PYNQ Z2 au routeur avec le câble Ethernet, puis à l'ordinateur avec le câble Micro-USB.

Figure 4.21 : carte PYNQZ2 connectée au pc

Ensuite nous utilisons la plateforme Jupyter afin de pouvoir accéder au contenu de la carte SD pour programmer le FPGA PYNQ Z2. Nous écrivons le mot de passe XILINX.

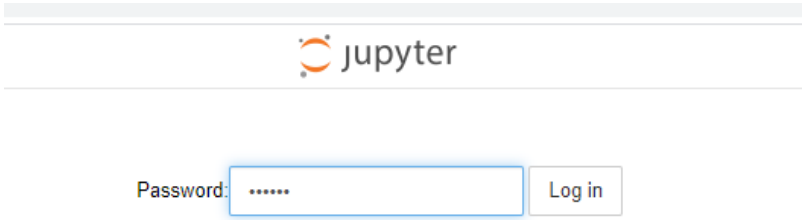


Figure 4.22 : JUPYTER

Afin de pouvoir programmer l'FPGA et réaliser l'accélération du modèle CNN implémenté, et ceci à travers la carte SD en utilisant la plateforme de Jupyter, nous devons ajouter les fichiers d'exécution et de configuration nécessaires, pour cela nous ajoutons un dossier qu'on a appelé « hw_bd » qui contient les deux fichiers .bit et .hwh (voir la figure ci-dessous). Nous ajoutons également les paramètres (du filtre et biais) du modèle CNN dans un dossier « parameters ». Nous ajoutons également un dossier « input_imgs » qui contient les images de test, ces dernières sont de taille 400x300. Finalement, nous ajoutons le fichier d'exécution HMC.ipynb écrit en python afin d'exécuter toutes les étapes sur FPGA pour réaliser l'implémentation. Ces étapes peuvent se résumer en ceci :

- Importer les packages de la plateforme PYNQ Z2,
- Entrer l'image de test (de taille 400x300 pixels),
- Charger l'overlay (detection_acc) créé par HLS de vivado,
- Charger en mémoire les paramètres du modèle CNN à savoir la taille du filter (weight) et le biais,
- Définir les données du réseau de neurone : définir sa taille,
- Prétraitement de l'image d'entrée : quantification et découpage en 5 parties afin de faire la reconnaissance des chiffres et des opérations mathématiques,
- Initialiser l'overlay : définir ses entrées et sorties et réaliser son adressage, puis réaliser la partie de contrôle, cette étape permet à l'FPGA de réaliser l'accélération du modèle CNN,
- Définir la partie calcul qui réalise l'opération et donne son résultat.

Figure4.23 : capture des fichiers ajoutés

Enfin, on exécute le fichier HMC.ipynb

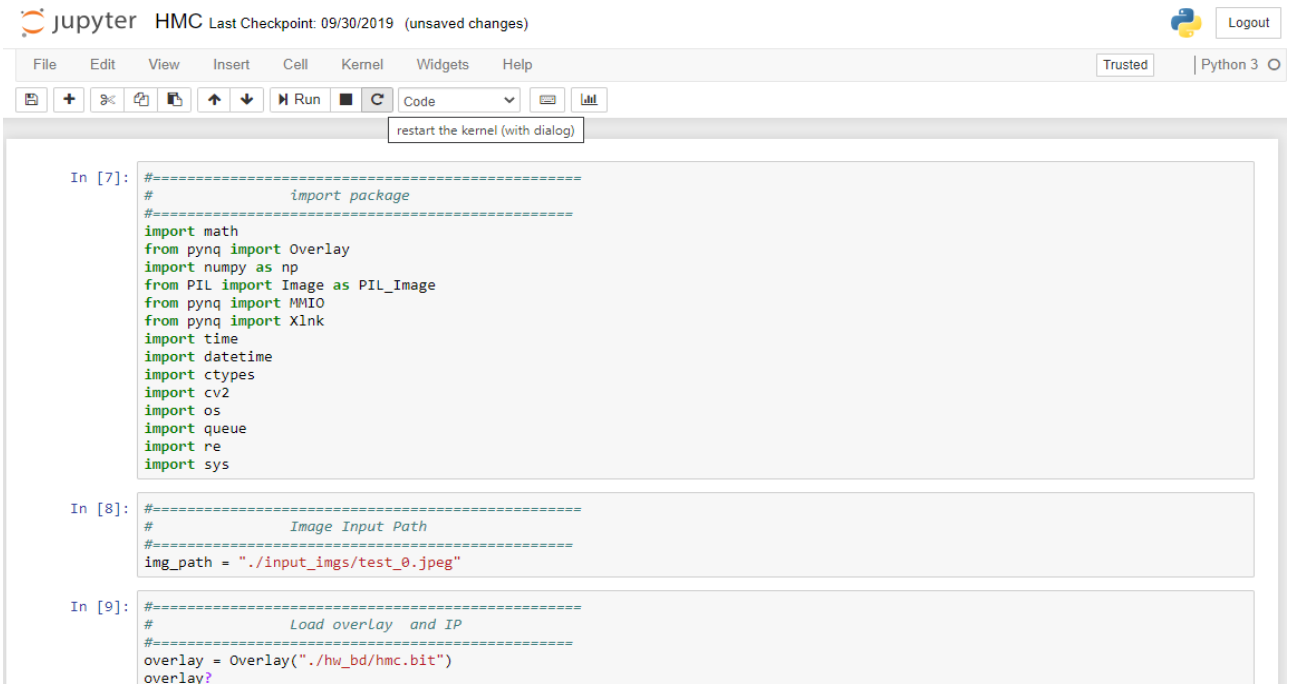
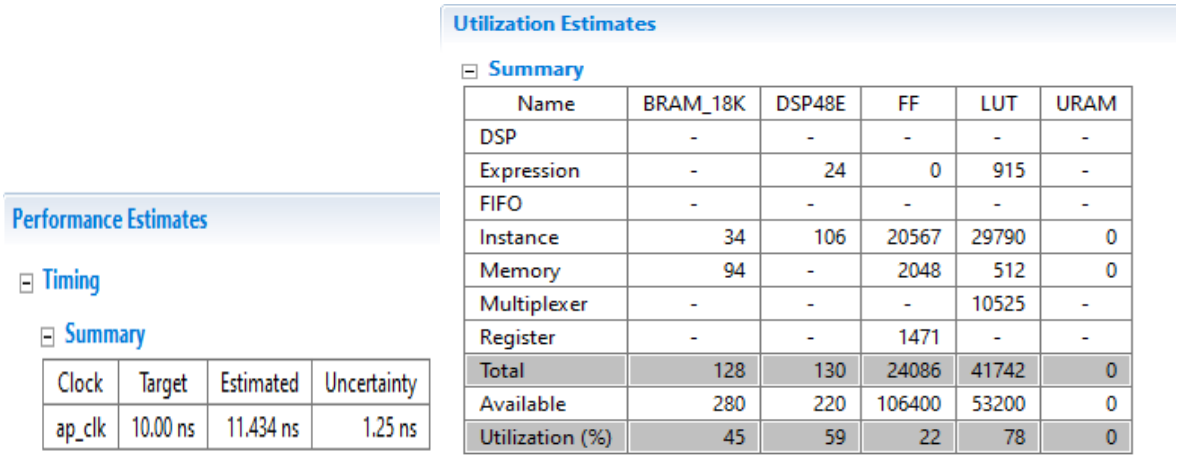


Figure 4.24 : L'exécution du fichier HMC

7. Les résultats :

Résultat de **la première étape :**



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	11.434 ns	1.25 ns

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	24	0	915	-
FIFO	-	-	-	-	-
Instance	34	106	20567	29790	0
Memory	94	-	2048	512	0
Multiplexer	-	-	-	10525	-
Register	-	-	1471	-	-
Total	128	130	24086	41742	0
Available	280	220	106400	53200	0
Utilization (%)	45	59	22	78	0

Figure 4.25 : Résultats de génération du core IP

Résultat de **la 2eme étape :**

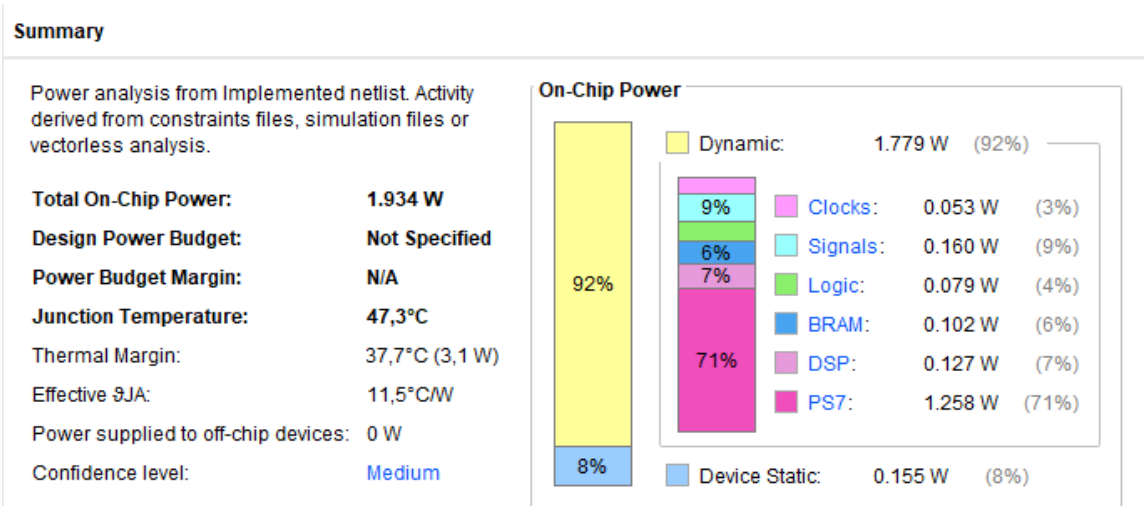


Figure 4.26 : consommation d'énergie

Résultat de **La 3eme étape :**

Nous testons le modèle avec une image d'entrée qui contient l'opération mathématique suivante : 1+2/4 écrite à la main (avec paint) et nous obtenons le résultat montré sur la figure suivante :

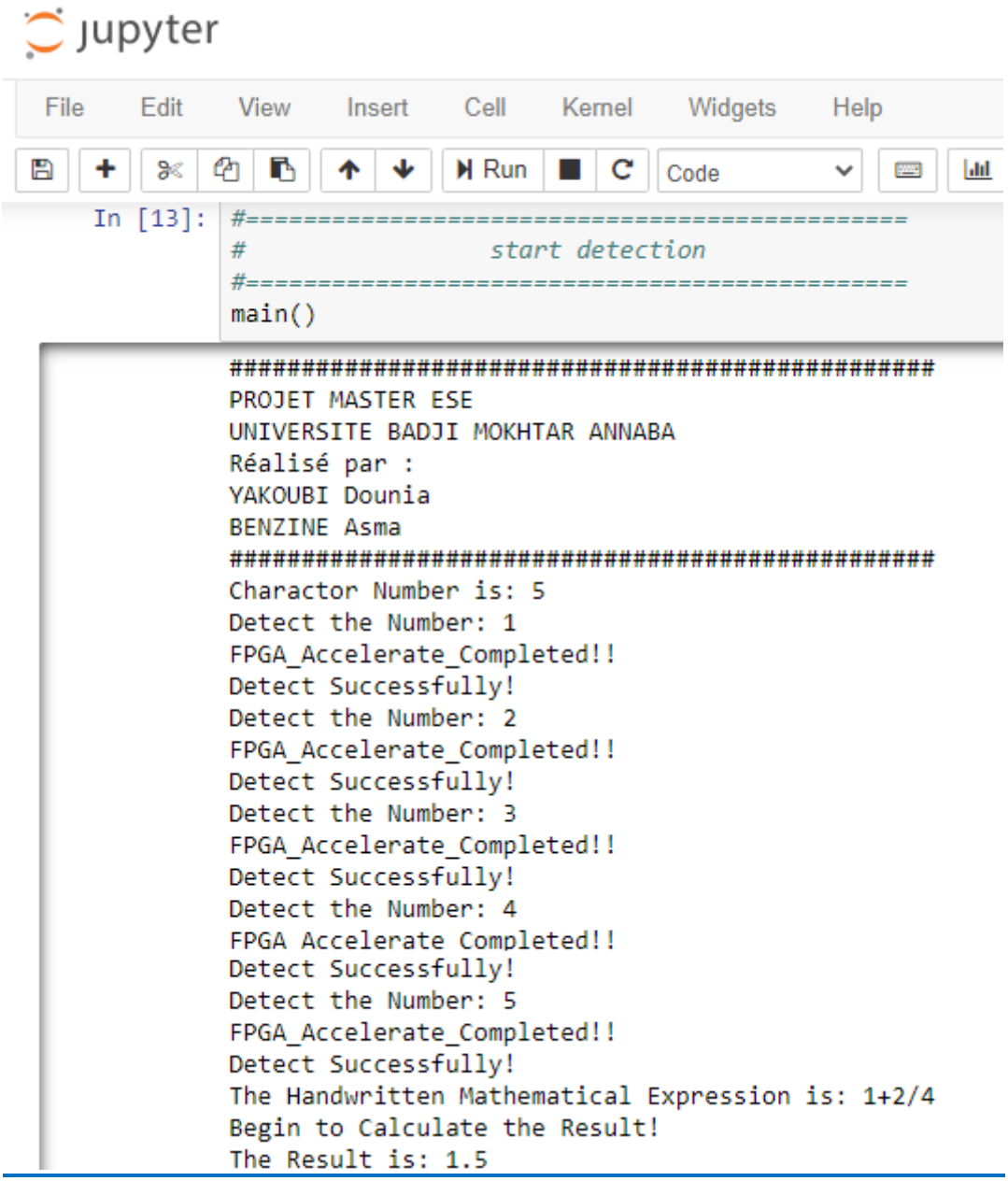


Figure 4.27 : résultat de l'exécution sur la carte PYNQ Z2

Conclusion :

Dans ce chapitre, nous avons implémenté sur une carte FPGA de type PYNQ Z2, un modèle CNN qui est capable de détecter des chiffres et des opérations mathématiques manuscrites, puis calculer le résultat de cette opération. Nous avons détaillé les étapes de cette implémentation avec les résultats obtenus pour chaque étape en utilisant le logiciel Vivado HLS 2019.2.

Conclusion général :

Afin d'obtenir de bonnes performances dans une calculatrice mathématique manuscrite, la reconnaissance d'images est une étape très importante et cruciale. Dans ce mémoire, nous avons développé une calculatrice mathématique manuscrite à base d'un réseau de neurones convolutif qui réalise une bonne détection et reconnaissance d'images. Ce travail s'articule autour de quatre chapitres :

- Dans le premier chapitre nous avons présenté une introduction générale sur les CNN et la reconnaissance d'image avec quelque exemple de l'industrie
- Dans le deuxième chapitre, nous avons fait l'étude des concepts des différentes phases d'apprentissage ainsi que les fonctions d'activation. Et nous avons détaillé les différentes couches du CNN de façon générale
- Dans le troisième chapitre nous avons abordé les éléments nécessaires afin de réaliser l'implémentation d'un réseau de neurones convolutif sur FPGA
- Dans le dernier chapitre, nous avons expliqué en détail chaque étape de la réalisation après cela, nous avons présenté les résultats obtenus.

Comme perspective, nous essaierons d'améliorer la détection des chiffres et opérations mathématiques ainsi que la précision de ce modèle CNN en ajoutant des couches de convolution afin de réduire d'éventuelles erreurs.

Références

[1] https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels

[2] BRAIKIA_BOUCHOUICHA. “Étude et implémentation d'un réseau de neurones convolutif CNN sur FPGA“ master universite badji mokhtar annaba , 2020.

[3] M-ELE.SY.E. “Reconnaissance d’images par les réseaux de neurones convolutionnels (CNN) “ master Université Mohammed Seddik BenYahia Jijel,2020

[4] https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5083336-decouvrez-les-differentes-couches-dun-cnn

[5] https://towardsdatascience.com/training-a-convolutional-neural-network-from-scratch-2235c2a25754

[6] Ivanovs, I. “Implementation analysis of convolutional neural networks on FPGAs” master projetct, Eindhoven University of Technology, 2019.

[7] ELLOUMI, Z. “Prédiction de performances des systèmes de Reconnaissance Automatique de la Parole”Thèse, Université Grenoble Alpes, 2019.

[8] https://www.amiq.com/consulting/2018/12/14/how-to-implement-a-convolutional-neural-network-using-high-level-synthesis/

[9] heotime Bollengier. Du prototypage à l’exploitation d’overlays FPGA. Systèmes embarqués. ENSTA Bretagne - École nationale supérieure de techniques avancées Bretagne, 2018. Français. ffNNT : 2018ENTA0003ff. fftel-02319236

[10] https://ichi.pro/fr/python-for-art-transfert-rapide-de-style-neuronal-a-l-aide-de-tensorflow-2-235189827559870

[11] Tong Geng*, Tianqi Wang*, Ang Li, Xi Jin, and Martin Herbordt FPDeep: Scalable Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters[9-12].

[12] C. Zhang, P. Li, G. Sun, et al., „Optimizing FPGA-based accelerator design for deep convolutional neural networks“, in Proceedings of the 2015 ACM/SIGDA International Symposium on FieldProgrammable Gate Arrays, ACM, 2015

[13] K.G.W. Goossens (TU/e) D. van den Heuvel "Implementation Analysis of Convolutional Neural Networks on FPGAs"

[14] C. Huang, S. Ni, and G. Chen. A layer-based structured design of CNN on FPGA. In 2017 IEEE 12th International Conference on ASIC (ASICON), pages 1037–1040, Oct 2017.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR, abs/1502.03167, 2015. 7

[16] Kevin Kiningham. Design and analysis of a hardware cnn accelerator. 2017. 15.

[17] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints, abs/1605.02688, May 2016.

[18] M. D. Youcef, « Deep Learning pour la classification des images », Mémoire de Master, Université Abou Bakr Belkaid Tlemcen, Faculté des sciences, Département informatique, 2017.

[19] Becker, M., Lippel, J., Zielke, T.: Gradient descent analysis: On visualizing the training of deep neural networks:. In: Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, 2019.

[20] H. Mouchere, C. Viard-Gaudin, R. Zanibbi, and U. Garain. ICFHR 2016 CROHME: Competition on recognition of online handwritten mathematical expressions. International Conference on Frontiers in Handwriting Recognition,2016.