RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

Département d'Electronique

Mémoire de Master

pour l'obtention du diplôme de Master

Implementation d'un Réseau de Neurones sur FPGA

Ghouthi BOUKLI HACENE

Sous la direction de
Dr. Rabah SADOUN

Présenté et soutenu publiquement le 18/06/2016

Composition du Jury :

Président	M. Mohamed TRABELSI	Prof.	ENP
Rapporteur	M. Rabah SADOUN	Dr.	ENP
Examinateur	M. Adel BELOUCHRANI	Prof.	ENP
Examinateur	M. Mourad ADNANE	Dr.	ENP

ENP 2016

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

Département d'Electronique

Mémoire de Master

pour l'obtention du diplôme de Master

Implementation d'un Réseau de Neurones sur FPGA

Ghouthi BOUKLI HACENE

Sous la direction de
Dr. Rabah SADOUN

Présenté et soutenu publiquement le 18/06/2016

Composition du Jury :

Président	M. Mohamed TRABELSI	Prof.	ENP
Rapporteur	M. Rabah SADOUN	Dr.	ENP
Examinateur	M. Adel BELOUCHRANI	Prof.	ENP
Examinateur	M. Mourad ADNANE	Dr.	ENP

ENP 2016

Remerciements

J'adresse mes remerciements aux personnes qui m'ont aidé dans la réalisation de ce projet de Master.

Je remercie en premier lieu Dr Rabah SADOUN d'avoir accepter d'être mon encadrant, et de m'avoir fait confiance et cru en moi.

Je tiens aussi à remercier particulièrement le professeur Vincent Gripon, sans qui je n'aurai pas découvert les réseaux de neurones, et qui m'a conseillé et aidé

Puis, Je remercie les membres du jury : Professeur Mohamed TRABELSI président du Jury, Pr. Adel BELOUCHRANI et Pr. Mourad ADNANE de m'avoir fait l'honneur d'examiner mon mémoire de Master.

ملخص

الهدف من هذا العمل هو تنفيذ الشبكة العصبية على بطاقة FPGA لدمج ملف المستخد في الهاتف الذكي. بعد تقديم بطاقاتFPGA ، سنشرع في القسم المخصص لخوارزميات الشبكة العصبية، بعدها نقوم سنتكلم خصوصا على الشبكة العصبية لـ GRIPON-BERROU والخوارزمية الخاصة بها. لاحقا تبدأ عملية تنفيذ الخوارزمية على بطاقة FPGA والتي سنشرحها بالتفصيل خلال هذه المذكرة.

الكلمات المفتاحية : شبكة عصبية. VHDL . FPGA. GBNN

Abstract

The objective of this work is to implement a neural network on FPGA in order to integrate the user's profile in the smartphone.

After a presentation of FPGAs, we will begin the section dedicated to neural network algorithms, then we'll talk particularly GBNN (GRIPON-BERROU neural network) and its algorithm.

Finally begins the implementation of this algorithm on an FPGA that will be detailed throughout this thesis.

Key words : FPGA, VHDL, C++, neural network, GRIPON-BERROU neural network.

Résumé

L'objectif de ce travail est d'implémenter un réseau de neurones dans un circuit FPGA afin d'intégrer le profil de l'utilisateur dans son smartphone.

Après une présentation des circuits FPGA, on entamera la partie dédiée aux algorithmes des RN, puis on parlera en particulier du GBNN (GRIPON-BERROU neural network) et de son algorithme.

Puis on entamera l'implémentation de cet algorithme dans un circuit FPGA qui va être détaillé tout au long de ce mémoire.

Mots clés : FPGA, VHDL, Langage C++, réseaux de neurones, réseau de neurone GRIPON-BERROU

Table des matières

Table des figures

Introduction

Nous utilisons tous les jours des smartphones pour faire des recherches sur Google, voir des vidéo sur Youtube ou encore réagir sur les réseaux sociaux. Ces applications gardent nos recherches afin d'apprendre ce qu'on aime le plus et en fonction de ça elles nous proposent des sujets qui peuvent nous intéressés. Cependant chacune de ces applications travaille sur des serveurs indépendamment des autres et donc Par exemple si on a tendance a suivre des tutorials sur Youtube, le serveur Google ne peut connaître une telle chose est donc nos recherches sur ce dernier ne seront pas guider par ce qu'on fait autre part.

Ceci nous amène à dire, et si c'est notre smartphone qui mémorise ces informations et nous guide durant nos recherches sur internet, il pourra connaître mémé notre liste de musique et réagir en conséquence, de plus du point de vue de l'utilisateur les données le concernant qui sont sur des serveurs posent le problème de la confidentialité. Ceci est d'autant plus vrai que la plupart des serveurs de calculs utilisés actuellement sont à l'étranger.

Viens s'ajouter a cela que des applications s'appuyant sur les profils d'utilisateurs. De telles applications fleurissent actuellement, mais chacune utilise son propre système déporté pour inférer les profils. L'utilisation de systèmes embarqués pour réaliser ces tâches permettrait une plus grande précision et un meilleur contrôle de la personnalisation des usages.

Très récemment, des auteurs ont montré qu'il était possible de compresser les réseaux de neurones profonds sans impacter de manière trop significative leur précision [1] [2]. Dans certains cas, une part importante des réseaux devient même binaire.

L'objectif étant de choisir un réseau de neurones de mémoires afin qu'il apprenne nos choix et préférences et permet a notre smartphone de gérer au mieux nos profils.

Le projet explicité prendra en compte le réseau de neurones GRIPON-BERROU GBNN [3] et cherche a l'implémenter sur une carte FPGA.

L'idée est d'abords de vérifier la validité du raisonnement de l'implémentation afin de concevoir le circuit équivalent de ce programme, avant de pouvoir traiter de l'application désirée. On s'intéresse donc dans ce mémoire uniquement à la partie

implémentation sur FPGA.

Ce rapport comporte, outre ce premier chapitre d'introduction, quatre autres chapitres dont la conclusion.

Le chapitre II introduit la FPGA, puis la carte et les outils de descriptions utilisés.

Le chapitre III présente les réseaux de neurones de manière général et le GBNN de manier particulière.

Le chapitre IV est un hybride des chapitres II et III où on fait la liaison entre eux en implémentant le GBNN sur la FPGA.

Chapitre 1

FPGA

Ce chapitre a pour objectif de présenter et familiariser le lecteur avec le système embarqué sur lequel le réseau de neurones sera implémenter, ainsi que les outils utilisés pour le décrire.

1.1 Une brève définition :

La FPGA (réseau de portes programmables ou Field Programmable Gate Array) [4] est un circuit intégré destiné à être configuré par l'utilisateur, d'où "programmable". La configuration d'une FPGA se fait généralement en utilisant un langage de description matérielle (HDL).

Les circuits FPGA sont constitués d'une matrice de blocs logiques programmables entourés de blocs d'entrée/sortie programmable. L'ensemble est relié par un réseau d'interconnexions programmable.

Les FPGA sont bien distincts des autres familles de circuits programmables tout en offrant le plus haut niveau d'intégration logique.

Il y a 4 principales catégories disponibles commercialement (figure 1.1) :

- Tableau symétrique.
- En colonne.
- Mers de portes.
- Les PLD hiérarchique.

La structure interne d'une FPGA de type matrice symétrique est décrite dans la figure 1.2. :

Les blocs logiques peuvent être configurés de telles sorte à ce qu'ils réalisent des fonctions combinatoires complexes soit elles ou simples.

Dans la plupart des FPGAs, les blocs logiques peuvent faire objet d'éléments de mémoire allant de simples bascules (Flip-Flop) jusqu'à des block mémoires complets

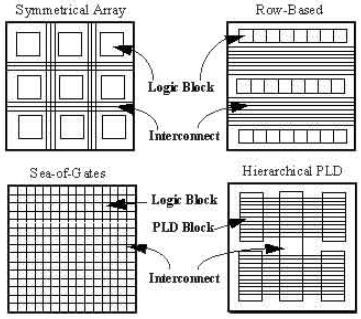

FIGURE 1.1: Les différentes classes de FPGA.

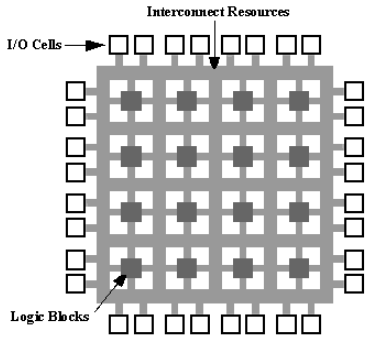

FIGURE 1.2: Structure interne d'un FPGA

(RAM, ROM, mémoire associative...).

1.2 Conception d'un système de base sur FPGA

La conception d'un système de base sur FPGA fait appel à deux approches :

1.2.1 Description HDL :

Cette approche utilise un langage de description matériel (HVDL ou Verilog) afin de définir les composants du système, les liaisons entre eux et leur fonctionnement interne. L'utilisateur est contraint de décrire la totalité de système lui-même à partir du zéro.

1.2.2 Utilisation des IP cores :

Un IP core (Semiconductor Intellectual Property Core) est un bloc logique propriétaire utilisé dans la conception d'un système sur FPGA ou circuit intégré ASIC. Un IP core représente une fonction simple soit elle ou complexe décrite, implémentée, testée et validée auparavant par son propriétaire et peut être donc intégrée directement dans le système éliminant le besoin de décrire la fonction en question manuellement.

La seconde méthode met en jeux l'utilisation des IP cores ayant des fonctions bien spécifiques et adéquates pour ensuite les relier et obtenir le système voulu. Contrairement à la première approche, celle-ci réduit le temps de conception et validation.

Dans le cadre d'une réalisation d'un système, l'utilisateur peut opter pour l'une des deux approches mais le plus souvent l'option hybride permet une flexibilité et une facilité de conception.

1.3 La carte utilisée :

AlteraDE1-SoC (figure 1.3)

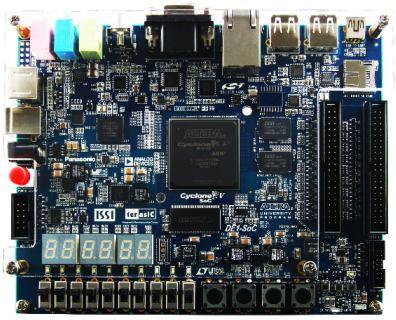

FIGURE 1.3: Image représentant la carte AlteraDE1-SoC

La DE1-SoC est une carte d'Altera qui implémente plusieurs fonctionnalités qui permettent à l'utilisateur de faire la conception de différents circuits allant d'un simple circuit à des projets multimédia complexes[5].

Les composants de la carte sont illustrés dans la figure 1.4.

La DE1-SoC a la particularité d'intégrer deux parties différentes à savoir :

Partie FPGA (Altera Cyclone® V SE 5CSEMA5F31C6N) qui comporte les éléments suivants :

- USB Blaster II for pour la programmation en mode JTAG.

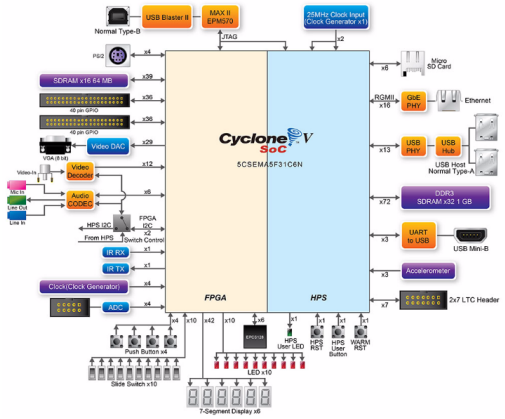

FIGURE 1.4: Les composants la carte AlteraDE1-SoC

- 64MB SDRAM (16-bit)
- 4 boutons poussoirs
- 10 interrupteurs
- 10 LEDs rouges
- 6 afficheurs 7 segments
- 4 horloges 50Mhz
- Audio CODEC 24-bit avec 3 prises Jack (Line in, Line out et microphone)
- VGA DAC (8) avec un connecteur VGA
- Décodeur TV (NTSC/PAL/SECAM) et entrée TV
- Un connecteur PS2
- Émetteur/Récepteur Infrarouge
- 2 fois 40 pins d'extension avec protection diode
- Convertisseur analogique/numérique
- Interface SPI 4 pins

Partie HPS (Hard Processor System) :

Contrairement à la PFGA, cette partie est figée – non configurable – et contient un processeur matériel

ARM Cortex-A9. Elle comporte les éléments suivants :

- Processeur Dual-core ARM Cortex-A9 MPCore 800MHz
- 1GB DDR3 SDRAM (32-bit bus de données)
- 1 Gigabit Ethernet PHY avec un connecteur RJ45
- 2 ports USB hôte, connecteur Type-A
- Lecteur carte Micro SD
- Accéléromètre (interface I2C + interruption)

- Adaptateur UART - USB, connecteur Type Mini-B
- 2 Boutons de réinitialisation matériel / logiciel
- 2 fois 7 connecteurs LTC

1.3.1 l'outil utilisé pour développer le système sur la DE1-Soc

Pour décrire un système sur notre carte FPGA on utilise l'outil Quartus.

Description de l'outil Quartus Altera Quartus II est un logiciel de conception de dispositif logique programmable produit par Altera. Quartus II permet l'analyse et la synthèse des conceptions de HDL, ce qui permet au développeur de compiler leurs conceptions, effectuer une analyse de timing, d'examiner les diagrammes RTL, simuler la réaction d'une conception à différents stimuli, et configurer le périphérique cible avec le programmateur. Quartus inclut une implémentation de VHDL et Verilog pour la description du matériel, l'édition visuelle des circuits logiques, et la simulation vecteur de forme d'onde (figure 1.5).

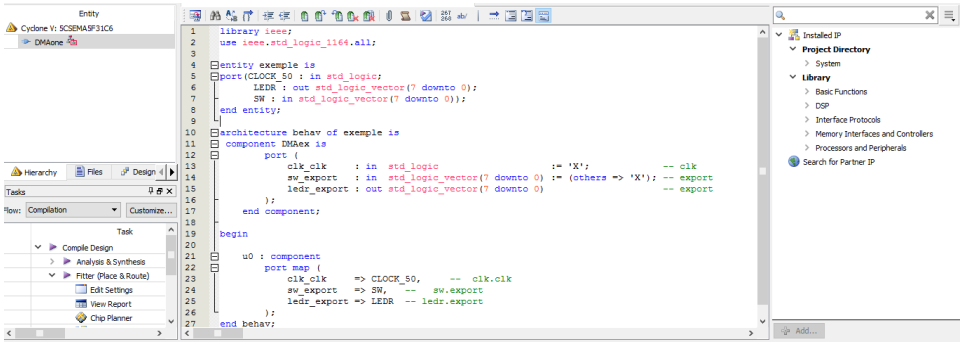

FIGURE 1.5: Exemple d'un programme sur Quartus.

L'outil Quartus admet aussi une description de système a l'aide des IPs core en faisant appel a Qsys.

Qsys est un outil intégré dans Quartus permettant la Description de système exécutant une fonction bien définit à l'aide des IPs core et de les relier entre eux afin qu'ils soient interconnectés décrivant ainsi un système prêt à être intégré sur la FPGA (figure 1.6).

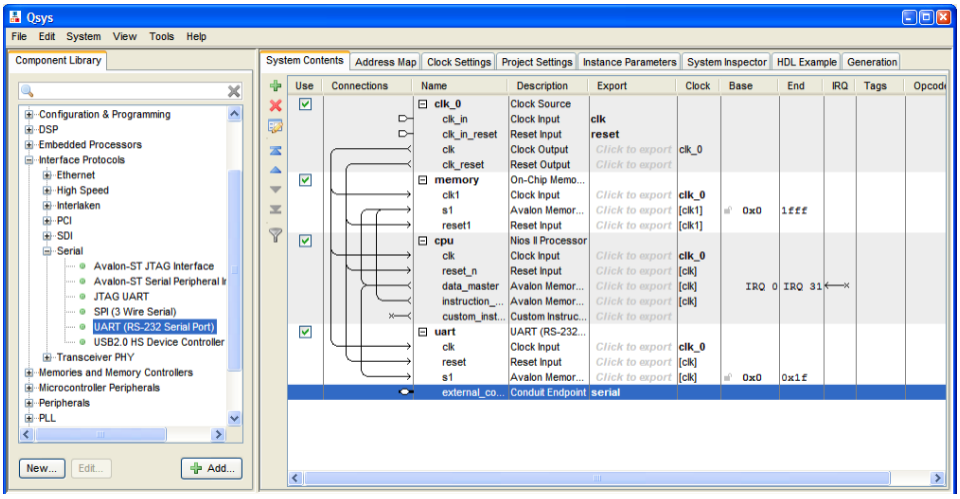

FIGURE 1.6: Exemple d'une description sur Qsys.

1.4 Conclusion

Ce chapitre a eu la FPGA comme principal sujet, car elle présente beaucoup d'avantage et permet une flexibilité considérable. Ce chapitre ainsi que le suivant sont les chapitres clé dans le projet car ils traitent les deux choses sur lesquels on se base à savoir la FPGA et les réseaux de neurones.

Chapitre 2

Réseaux de neurones

Ce chapitre présentera les réseaux de neurones artificiels en général, leur philosophies ainsi que leur fonctionnement, puis on abordera le réseau de neurone GRIPON-BERROU (GBNN) qui est choisit grace a ses performances [3] et qu'on cherche à implémenter.

2.1 Présentation des réseaux de neurones

Dans le domaine d'apprentissage automatique et des sciences cognitives, les réseaux de neurones artificiels (ANN) sont une famille de modèles inspirés par les réseaux biologiques neuronaux (les systèmes nerveux central des animaux, en particulier le cerveau) qui sont utilisées pour estimer et approximer des fonctions qui peuvent dépendre d'un grand nombre d'entrées et sont généralement inconnues [6].

Les réseaux neuronaux artificiels sont généralement spécifiés à l'aide de trois choses suivantes :

- **L'architecture :** elle spécifie quelles variables sont impliquées dans le réseau et leurs relations à des fins topologiques, exemple les variables impliquées dans un réseau de neurones pourraient être les poids des connexions entre les neurones, ainsi que les activités des neurones.
- **Règle d'activité :** La plupart des modèles de réseaux neuronaux ont une courte dynamique suivant l'échelle de temps : les règles locales définissent la manière dont les activités des neurones changent.Typiquement, la règle de l'activité dépend des poids (les paramètres) dans le réseau.
- **Apprentissage :** La règle d'apprentissage spécifie la façon dont les poids du réseau de neurones changent avec le temps. Cet apprentissage est généralement considéré comme ayant lieu sur une échelle de temps plus longue que l'échelle

de temps de la dynamique sous la règle d'activité. Habituellement, la règle d'apprentissage dépendra des activités des neurones. Elle peut aussi dépendre de la valeur des valeurs cibles fournies par un enseignant et de la valeur actuelle des poids.

Par exemple, un réseau de neurones pour la reconnaissance de l'écriture manuscrite est définie par un ensemble de neurones d'entrée qui peuvent être activées par les pixels d'une image d'entrée. Après avoir été pondérés et transformés par une fonction (déterminée par le concepteur du réseau), les activations de ces neurones sont ensuite transmises à d'autres neurones. Ce processus est répété jusqu'à ce que, le neurone de sortie détermine quel caractère a été lu est activé.

Un réseau neuronal artificiel est un groupe de noeuds inter-connectés, semblable à un vaste réseau de neurones dans le cerveau. Dans la figure 2.1, chaque noeud circulaire représente un neurone artificiel et une flèche représente une liaison entre la sortie d'un neurone à l'entrée d'un autre. comme le montre la figure 3.1.

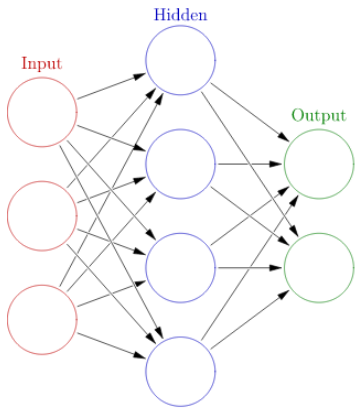

FIGURE 2.1: Structure d'un réseau de neurone

Un réseau de neurones est en général composé d'une succession de couches (couche d'entrée, les couches cachées et la couche de sortie) dont chacune prend ses entrées sur les sorties de la précédente. Chaque couche (i) est composée de Ni neurones, prenant leurs entrées sur les Ni-1 neurones de la couche précédente. À chaque neurone sont associés des poids, de sorte que les entrées sont multipliées par ce poids, puis additionnés, ce qui est équivalent à multiplier le vecteur d'entrée par une matrice de transformation [7].

Mettre les couches en cascade dans un réseau de neurones reviendrait à mettre en cascade plusieurs matrices de transformation et pourrait se ramener à une seule matrice, produit des autres, s'il n'y avait à chaque couche, la fonction de sortie

qui introduit une non linéarité à chaque étape. Ceci montre l'importance du choix judicieux d'une bonne fonction de sortie et donc un réseau de neurones dont les sorties seraient linéaires n'aurait aucun intérêt, car les couches cachées sont utilisées spécialement pour faire des calculs complexes remplissant une fonction particculière (comme par exemple la convolution).

Le calcul se fait dans chaque neurone où ce dernier calcule la somme de ses entrées puis cette valeur passe à travers la fonction d'activation pour produire sa sortie, comme le montre la figure 2.2.

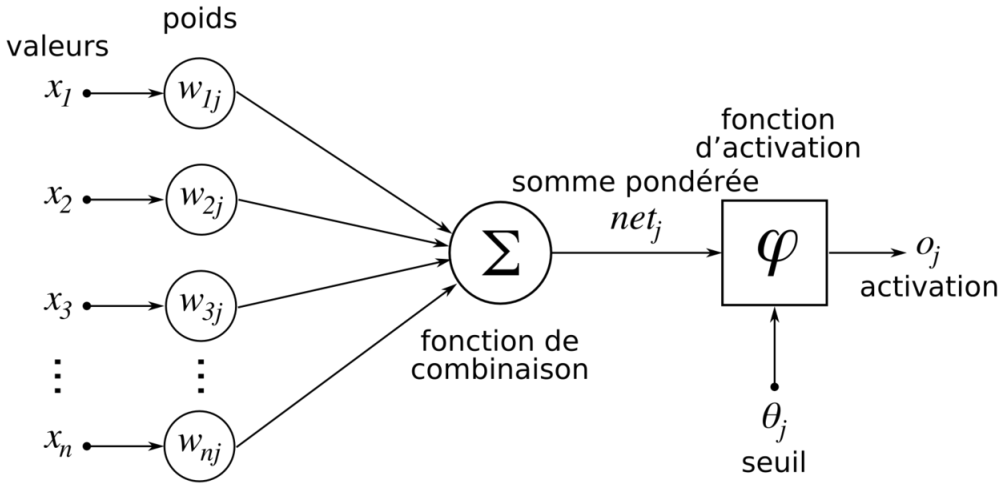

FIGURE 2.2: Exemple de calcul d'un neurone

Fonction d'activation La fonction d'activation a pour principal rôle d'introduire la non-linéarité dans le fonctionnement du neurone. La fonction présente 3 intervalles :

- inférieur au seuil, neurone non actif.
- aux alentours du seuil, phase de transition.
- superieur au seuil, neurone actif.

L'exemple classique de la fonction d'activation est la fonction Sigmoïde qui est :

$$\frac{1}{1+\exp^{-x}}$$

2.2 Réseau de neurone GBNN

Le réseau de neurone GBNN [3] est un réseau de neurone de mémoire. le réseau apprend des messages reçus.

Les messages reçu pour l'apprentissage se composent de 8 nombres ayant des valeurs entre 0 et 255.

La structure de ce réseau est différente de celle présentée précédemment. il est composé de 8 regroupements contenant 256 neurones et chaque nombre représente un neurone du regroupement. Donc pour un seul message un neurone est activé par regroupement, puis sont reliés entre eux en mettant les poids qui correspondent aux liaison a 1. une fois la phase d'apprentissage terminée on obtient une structure du réseau telle explicitée dans la figure 2.3.

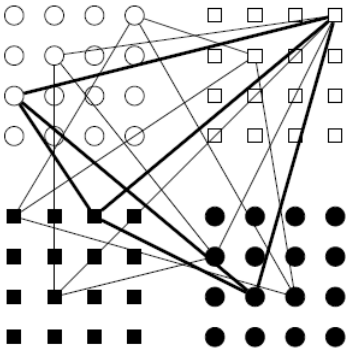

FIGURE 2.3: Exemple de structure du réseau avec 4 regroupements 16 neurones par regroupement

Une fois la phase d'apprentissage terminée, on souhaiterait tester la structure obtenue. Pour le faire on prend un message déjà appris qu'on efface sa moitié et on l'injecte au niveau du réseau.

4 neurones seront activés dans 4 regroupements différents et notre but est de retrouver les 4 autres neurones dans les 4 regroupements restant afin d'obtenir le message.

Pour cela on passe par deux étapes de décodage dont le décodage local est global.

2.2.1 Décodage local

Le décodage est local au regroupement, cela veut dire qu'il se passe au niveau de chaque regroupement indépendamment des autres.

Son principe est de prendre au sein du même regroupement tous les neurones, puis injecter à chaque neurone 4 entrées correspondantes aux 4 regroupements obtenus par le message à moitié effacé (l'entrée est égale à 1 si le neurone en question est relié au neurone activé dans le regroupement correspondant sinon elle est égale à -1).

Les entrées de chaque neurone sont sommées et le résultat est mit à 0 si ce dernier est négatif comme explicité dans la figure 2.4.

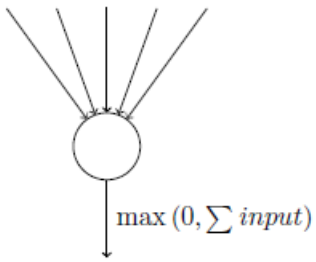

FIGURE 2.4: Un model de neurone

On garde que les neurones où la somme est maximale faisant ainsi un décodage local .

2.2.2 Décodage global

Le décodage global vient tout de suite après le décodage local, où il prend les neurones résultant et teste si chacun d'eux a 3 liaisons avec 3 neurones des 3 regroupements auquels il n'appartient pas et aucun neurone ne fut activé au moment de la réception du message a moitié effacé.

2.2.3 Validation de l'algorithme

Pour valider l'algorithme un programme en C++ fut développé pour tester la structure du réseau et plus important encore, avoir le taux d'erreur commise lors de la récupération du message en fonction du nombre de messages appris.

Création des messages On crée M des messages aléatoirement grâce à la fonction suivante :

```
void createMessage()
{
      int i,j;

     srand(time(NULL));

     for(j=0;j<M;j++)
     {
          for(i=0;i<8;i++)
          {
               Message[j][i]=rand() % 256;
          }

          //afficher(Message[j]);
     }
}
```

Création des neurones On commence par déclarer une classe qui fera référence au neurone et qui contient :

- l'id du neurone
- le regroupment auquel il appartient (sous forme de nombre)
- le nombre de connexions qu'il a
- les adresses des neurones avec qui il est relié.

```
class neuron
{
private:
    unsigned char id;
    int clust_number;
    int connections;
    neuron** NeCo;

    public:
        neuron(unsigned char id, int clust_number);
        neuron();
        void connect(neuron* N);
        unsigned char getid();
        int getclust_number();
        int getConnections();
        void setConnections(int a);
        neuron** getNeco();
        void setNeco(neuron** inNeco);

};
```

Phase d'apprentissage Dans cette phase on prend les 8 parties du message et on active les neurones avec les ids correspondant puis on les relit entre eux. la fonction C++ responsable de cette partie est :

```
void learn()
{

    int i,j,k;
    for(i=0;i<M;i++)
    {
        for(j=0;j<8;j++)
        {
            //center =Network[j][Message[i][j]];
            for(k=j+1;k<8;k++)
            {
                //neighbour=Network[k][Message[i][k]];
                if(!isTheSame(&Network[j][Message[i][j]],&Network[k][Message[i][k]])){
                Network[j][Message[i][j]].connect(&Network[k][Message[i][k]]);
                Network[k][Message[i][k]].connect(&Network[j][Message[i][j]]);
                }

            }
        }
    }
}
```

Phase de décodage local La fonction responsable de cette phase est "detect" qui fait le decodage en utilisant les étapes suivantes :

- prendre la moitié du message et localisé les 4 neurones dans les 4 regroupements.
- trouver pour chaque neurone activé, les neurones avec lesquels il est relié et qui appartiennent aux autres regroupements qui correspondent à la partie effacée.
- garder uniquement les neurones communs.

Phase de décodage global La fonction responsable de cette partie est : "global-Detected". Exécutant l'algorithme explicité précédemment cette fonction est exécutée 4 fois afin de faire des itérations et améliorer le résultat.

Phase de calcul d'erreur cette fonction prend le nombre de neurones obtenus après le décodage global auquel elle retranche 4 (car avoir 4 neurones veut dire que le message a bien été récupéré sans erreur). on refait la même opération 100 fois et on somme les nombres obtenus puis on les divise par 100.

le tableau 2.1 contient les résultats obtenus en fonction du nombre de messages appris, et la figure 2.5 explicite ces résultats sous forme de courbe.

2.3 Conlusion

Le réseau de neurones est un outil permettant à partir de certaines données d'apprendre comment elles sont reliées et quelles sont les relations les décrivant puis

Nombre de message	erreur
0	0
10000	0.02
15000	0.08
20000	0.21
22000	0.29
25000	0.51
28000	0.67
30000	0.774
31000	0.81
32000	0.855
35000	0.89
40000	0.938148
50000	0.96

TABLE 2.1: Erreur en fonction du nombre de messages appris

appliquer ces relations sur de nouvelles données.

Le réseau de neurones GBNN permet de mémoriser des données puis les retrouver même si elles sont endommagées avec des erreurs négligeables pour un nombre de messages appris inférieur a 15000. Il peut servir pour corriger les erreurs dues à la transmission de données comme il peut faire en sorte d'augmenter la vitesse de communication en n'envoyant qu'une partie du message.

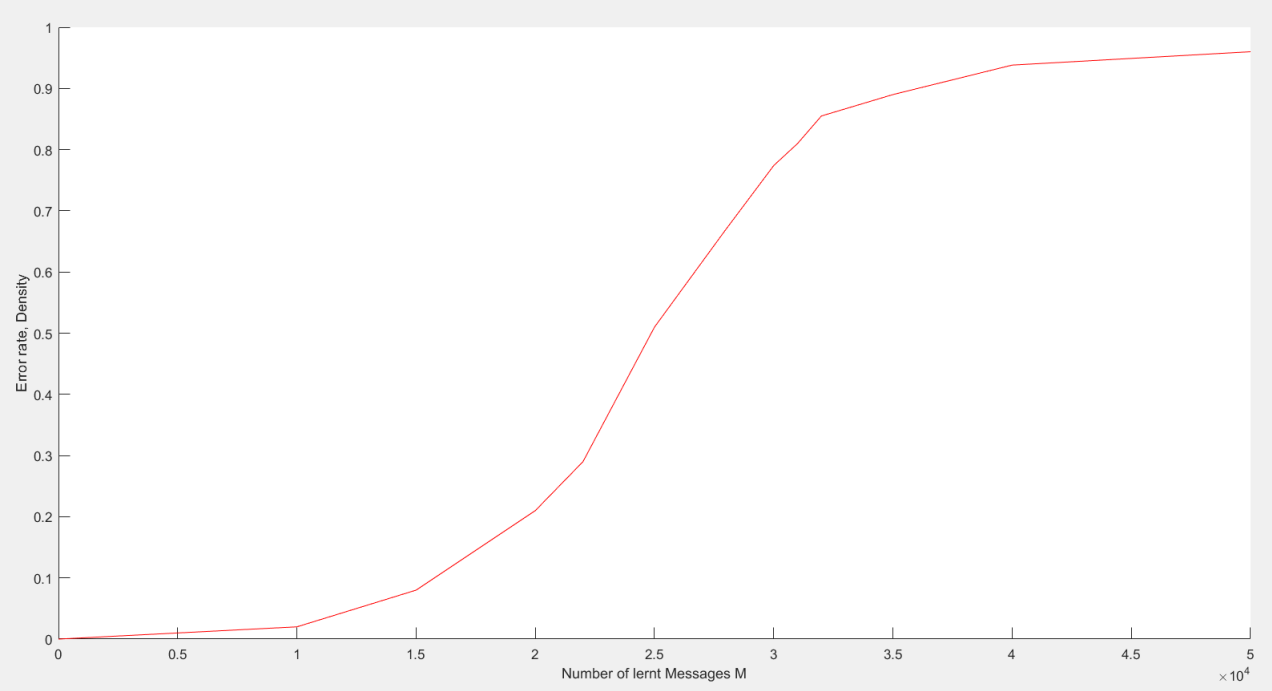

FIGURE 2.5: Représentation de l'erreur en fonction du nombre de messages appris

Chapitre 3

Implémentation du réseau de neurones GBNN sur FPGA

Dans ce chapitre nous allons nous intéresser à la façon d'intégrer le réseau de neurones GBNN dans la carte FPGA, le principe utilisé ainsi que les ressources nécessaires pour une telle implémentation. L'exemple discuté aura 8 regroupements dans lesquelles on trouve 64 neurones, et est décrit en VHDL.

3.1 Discussion et implémentation de la phase d'apprentissage

L'apprentissage se base essentiellement sur un nombre de messages générés aléatoirement. Cette étape est indépendante de la partie HDL, où on prend un fichier contenant les messages et on les injecte directement dans des bloques de mémoires.

Chaque message contient les neurones actifs et donc c'est un message binaire de 8 fois 64 bits où chaque bit représente un neurone, et quand ce bit est à 1 le neurone est actif.

On a un bloc mémoire pour chaque regroupement et chaque bloc contient 64 lignes mémoires ou chaque ligne mémoire a une taille de 512 bits, et qui sont initialisées par des 0.

```
type matrix is array (0 to 63) of std_logic_vector(511 downto 0);

signal memory1,memory2,memory3,memory4,memory5,memory6,memory7,memory8: matrix;
```

Pour chaque message on prend le iémé 64 bits, et l'id du neurone actif à cet

endroit représente la ligne du iémé bloc, où le contenu de cette ligne est utilisé dans un OU logique avec le message et le résultat et mit à la ligne en question.

Faisant ceci pour toutes les parties du message et pour tous les messages on obtient des blocs mémoires représentant la phase d'apprentissage avec les neurones et leur liaisons.

Puisque cette phase se fait une seule fois, on peut faire en sorte de prendre un fichier qui contient les données à mettre sur les blocs mémoires et on lit directement le contenu du fichier pour le placer dans la mémoire.

```
 FILE f : TEXT;
 constant filename : string :="output.txt";
 VARIABLE L : LINE;
 variable i : integer:=0;
 variable b: std_logic_vector(511 downto 0);
begin
File_Open (f,FILENAME, read_mode);
      for i in 0 to 63 loop -- the first cluster
      readline (f, l);
      read(l, b);
      memory1(i) <= b;
      end loop;
```

3.2 Discussion et implémentation du décodeur local

dans notre implémentation on assume connaître la partie effacée qu'on représente grâce à un vecteur de 8 bits ou chaque bit nous informe si le regroupement correspondant reçoit la partie effacée du message.

On prend chaque partie du message qu'on injecte dans le bloc mémoire correspondant et grâce a l'id qui se trouve dans cette partie on obtient un vecteur de 512 bits de '1' si cette partie correspond à la partie effacée sinon le résultat correspond aux données décrivant le neurone ayant cet id (les neurones avec lesquels il est relié).

```
Message_retrieved(0)<=(others=>('0')) when EraseFlag(0)='0' else
                        memory1(to_integer(unsigned(MessageIn(5 downto 0))));

Message_retrieved(1)<=(others=>('0')) when EraseFlag(1)='0' else
                        memory1(to_integer(unsigned(MessageIn(11 downto 6))));

Message_retrieved(2)<=(others=>('0')) when EraseFlag(2)='0' else
                        memory1(to_integer(unsigned(MessageIn(17 downto 12))));

Message_retrieved(3)<=(others=>('0')) when EraseFlag(3)='0' else
                        memory1(to_integer(unsigned(MessageIn(23 downto 18))));

Message_retrieved(4)<=(others=>('0')) when EraseFlag(4)='0' else
                        memory1(to_integer(unsigned(MessageIn(29 downto 24))));

Message_retrieved(5)<=(others=>('0')) when EraseFlag(5)='0' else
                        memory1(to_integer(unsigned(MessageIn(35 downto 30))));

Message_retrieved(6)<=(others=>('0')) when EraseFlag(6)='0' else
                        memory1(to_integer(unsigned(MessageIn(41 downto 36))));

Message_retrieved(7)<=(others=>('0')) when EraseFlag(7)='0' else
                        memory1(to_integer(unsigned(MessageIn(47 downto 42))));
```

Le code développé en C++ utilisait le principe des neurones communs pour faire l'opération du décodage local. ce principe au niveau du VHDL se traduit par un ET logique entre tous les résultats de l'étape précédente.

```
MessageOut<= message_retrieved(0) and message_retrieved(1) and message_retrieved(2) and
             message_retrieved(3) and message_retrieved(4) and message_retrieved(5) and
             message_retrieved(6) and message_retrieved(7);
```

3.3 Discussion et implémentation du Décodage global

On commence par prendre le résultat du local decoding et on met un xor avec le message reçu (qui est effacé partiellement et on admet que les parties effacées sont représentées par un vecteur de 64 bits contenant que des 0).

```
Message_decoded<=MessageIn_decoded xor LocalDecodingOut;
```

On prend chaque 64 bits (chaque regroupement) et on fait un OU logique entre les 64 bits du même vecteur, si le résultat est égal a 0 (le signal c dans le code) cela voudrait dire que cette partie n'est pas effacée est donc pas besoin d'appliquer le décodage global sur cette partie.

```
      -- if c=0 then this 64 bits corresponding to non erased part
v0: logicalOR64 port map(MessageIn_decoded(63 downto 0),c(0));
v1: logicalOR64 port map(MessageIn_decoded(127 downto 64),c(1));
v2: logicalOR64 port map(MessageIn_decoded(191 downto 128),c(2));
v3: logicalOR64 port map(MessageIn_decoded(255 downto 192),c(3));
v4: logicalOR64 port map(MessageIn_decoded(319 downto 256),c(4));
v5: logicalOR64 port map(MessageIn_decoded(383 downto 320),c(5));
v6: logicalOR64 port map(MessageIn_decoded(447 downto 384),c(6));
v7: logicalOR64 port map(MessageIn_decoded(511 downto 448),c(7));
```

Si c=1 alors on teste chaque neurone si sa valeur est égale à 1 on prend le vecteur de 512 bits dans la mémoire correspondant à ce neurone et on met un ET logique avec le résultat du décodage local, puis on fait un OU logique sur les 64 bits de chaque regroupement, on obtient un signal de 8 bits et finalement on fait un ET logique entre ces 8 bits. Si le résultat est égale à 0 cela voudrait dire que ce neurone n'appartient pas au message car il existe un regroupement où tous les neurones activés après le décodage local ne sont pas reliés au neurone en question et donc il faut le mettre à 0.

3.4 Conclusion

Ce chapitre a traité l'implémentation du réseau de neurones GBNN sur la carte FPGA. Il a permis aussi de proposer une nouvelle vision des algorithmes permettant de faciliter la description du réseau. Cependant l'évaluation des ressources qu'utilise la carte pour implémenter ce réseau n'est pas encore prise en considération car d'autres blocs s'ajouteront par la suite afin de donner un fonctionnement complet telle la prise en charge de l'erreur au niveau du message qui ne provoque pas d'effacement.

chapter*Conclusion Notre projet de fin d'étude a évolué suivant plusieurs phases ou chaque étape représentait une progression considérable et qu'on explicite dans le diagramme de Gantt suivant. figure 3.1 :

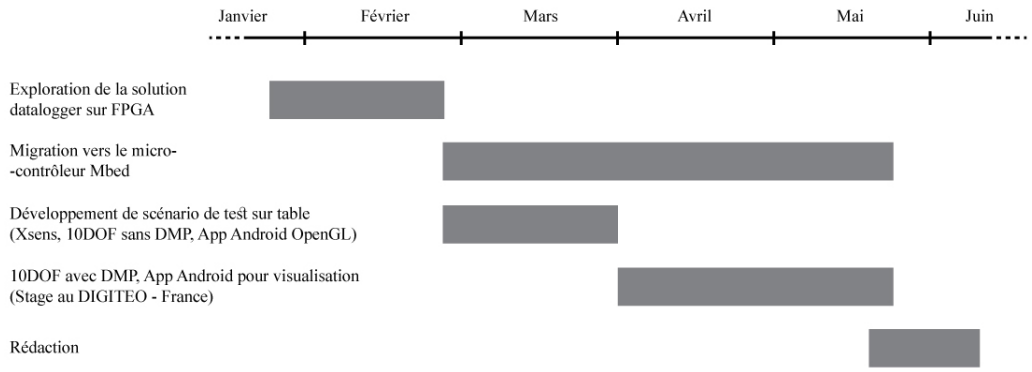

FIGURE 3.1: Diagramm de Gantt

Le projet VIROLO++ est toujours en cours de développement, il vise à mieux comprendre le comportement du véhicule deux roues ainsi que son conducteur dans des situations spécifiques (telle la prise de virage), et à proposer une architecture système optimale pour gérer au mieux les fonctionnalités qu'on souhaite apporter. Etant un projet ambitieux, on vise à l'améliorer sans cesse afin d'obtenir un produit fini facilement embarquable sur différents type de véhicule deux roues répondant ainsi à un besoin exprimé par la gendarmerie nationale française, et donc comme tout projet de cette ampleur, il doit avoir des perspectives lui permettant de s'adapter et de s'imposer par-rapport aux autres produit. Ses perspectives à cout-terme concerne spécialement la partie Visuelle, à savoir l'application Android et qui sont : - Ajouter au niveau des paramètres un nouvel identifiant de trame CAN avec son format (i.e. le format des données, le nombre de bits sur lequel une donnée est codée). - Une charte personnalisable, ou on peut ajouter des graphes selon notre besoin pour faire la comparaison. - Afficher dans Google Maps les informations correspondantes a un point du trajet effectué (vitesse, angle de roulis, accélération, direction, ...etc.) lors d'un clique sur le point. - Traçage de la trajectoire sur des cartes IGN avec Geoportail (à la place de Google Maps). Les perspectives à moyen terme consistent à proposer une solution complète et fonctionnelle qui permet aux moto-écoles de former les conducteurs et d'évaluer objectivement leurs comportements. Les perspectives à long terme résident dans la prévention d'accident afin de déclencher le système quelques secondes avant l'impact permettant d'amortir le choc et donc sauver des vies, ainsi que l'option d'autopilote peut être abordé en suivant le développement du projet.

Bibliographie

[1] Y. Chen. *Compressing convolutional neural networks.* "arXiv preprint arXiv :1506.04449". 2015.

[2] V. Gripon G. Soulié et M. Robert. *Compression of deep neural networks on the fly.* `http://arxiv.org/abs/1509.08745`.

[3] IEEE Vincent Gripon Student Member et IEEE Claude Berrou Fellow. "Sparse neural networks with large learning diversity". In : *SD Times (BZ Media LLC)* (2012).

[4] *FPGA Architecture for the Challenge.* `http://proxacutor.free.fr/glossaire.htm#logique`.

[5] *DE1 User Manual.* 2006.

[6] Bhadeshia H. K. D. H. *Neural Networks in Materials Science.* `http://www.msm.cam.ac.uk/phase-trans/abstracts/neural.review.pdf`. 1999.

[7] Frank Rosenblatt. *The Perceptron : probabilistic model for information storage and organization in the brain.* "Psychological Review, 65 :386-408". 1958.