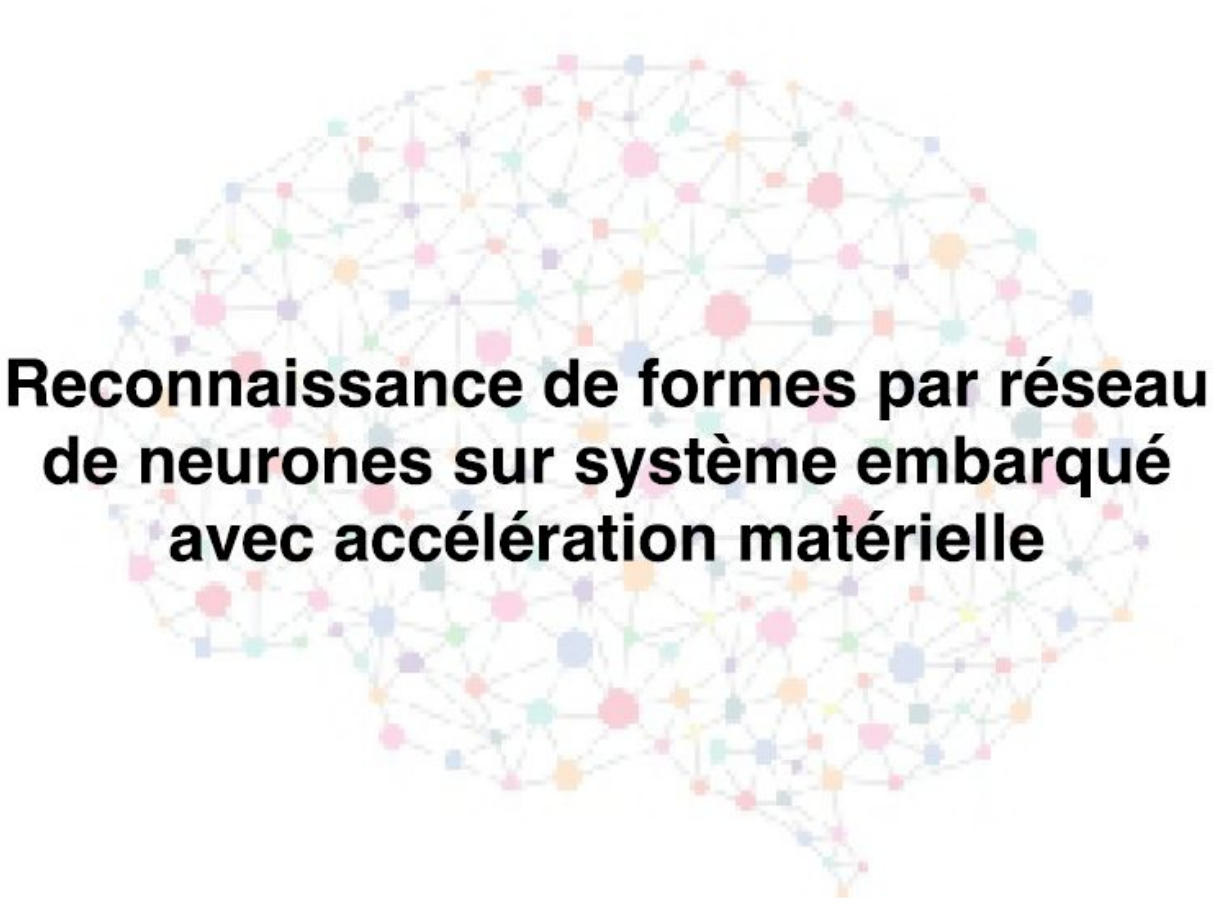
PX 504 - Projet d'innovation



BARRIÈRE Louka - CANET Gauthier

LE DONGE Romain - MORICEAU Alexandre

Sommaire

Remerciements

Nous tenons à remercier notre chargé de projet monsieur Vincent Beroulle pour le soutien qu'il nous a apporté sur le plan technique comme sur le plan de l'organisation du projet

.

I) Introduction

L'intelligence artificielle est aujourd'hui présente dans un nombre grandissant de systèmes embarqués : systèmes de navigation dans nos véhicules, assistants personnels, reconnaissance vocale et faciale dans nos smartphones, véhicules autonomes….
Cela a été rendu possible par les progrès immenses qui ont été réalisés dans le domaine des composants électroniques (processeurs, cartes graphiques…) et par l'accès croissant à des données massives (notamment grâce à l'essor des objets connectés) qui ont permis de développer et d'améliorer les technologies d'apprentissage automatique (machine learning). L'une de ces technologies, les réseaux de neurones artificiels, connaît un regain d'intérêt auprès des chercheurs (il fait par l'exemple l'objet d'une offre de stage du CEA : ref 3387242) comme des géants industriels comme IBM ou encore Google.

Dans le cadre du projet d'innovation nous avons souhaité nous aussi nous intéresser aux réseaux de neurones. C'est la raison pour laquelle notre projet consistait d'une part à utiliser cet technique d'apprentissage automatique en l'intégrant dans un système embarqué pour reconnaître des formes réalisées à l'aide d'un accéléromètre. D'autre part, afin de proposer quelque chose d'innovant, nous avons souhaité en plus de cela démontrer, via une preuve de concept, qu'il est possible d'optimiser cette technique d'apprentissage en utilisant un FPGA.

Les trois objectifs du projet étaient donc les suivants :

1) comprendre et adapter un réseau de neurones à un besoin spécifique;

2) réaliser un démonstrateur permettant, via l'utilisation d'un réseau de neurones artificiels, de reconnaître des formes basiques à partir de mouvements;

3) réaliser une accélération matérielle de notre système embarqué.

Dans ce rapport nous commencerons par donner le contexte technique nécessaire à la compréhension de nos réalisations puis nous présenterons l'architecture générale que nous avons mise en place puis suivie tout au long du projet. Nous présenterons ensuite les travaux effectués pour chacune des fonctionnalités de notre démonstrateur avant d'enchainer avec l'intégration de ces parties et les résultats qui en ont découlé. Enfin nous conclurons ce rapport avec le bilan de nos résultats vis à vis des objectifs du projet ainsi qu'avec les apports personnels que nous a donné ce projet d'innovation.

II) Présentation du contexte

Les réseaux de neurones artificiels connaissent un fort succès actuellement car ce sont des algorithmes flexibles donnant de très bons résultats. Cette méthode d'apprentissage tente de modéliser le fonctionnement du cerveau humain et donc des cellules qui le composent : les neurones. Un neurone biologique est une cellule qui comporte quatres éléments principaux : une dendrite, un coeur qui est une cellule somatique, un axone et des synapses. Le fonctionnement du neurone est le suivant : il reçoit des signaux électriques par ses dendrites qui sont ensuite propagés à la cellule somatique qui fait faire une "somme" des signaux. La somme des signaux est propagée sur l'axone qui va ensuite, via les synapses, être connectée à d'autres dendrites d'autres neurones, etc... Dans les réseaux de neurones artificiels on va essayer de reproduire ce comportement. Un neurone artificiel peut être représenté par une somme d'entrées pondérées à travers une fonction d'activation, dans notre cas une fonction sigmoïde. Le schéma présenté en figure 1 représente de gauche à droite : une vue d'un neurone , la fonction d'activation sigmoïde et un assemblage de neurones a plusieurs **couches**.

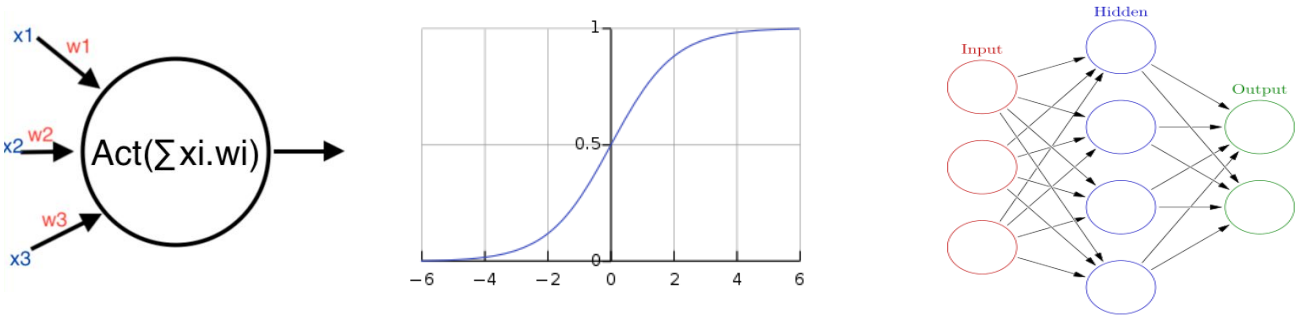


Figure 1 : Fonction d'activation d'un neurone, fonction sigmoïde et exemple d'architecture d'un réseau de neurones

D'un point de vue mathématique un neurone est défini par une fonction, mais si l'on prend un peu de hauteur, un réseau de neurones entier est défini premièrement par des matrices qui représentent ces synapses et deuxièmement par des produits de matrices qui lient les synapses d'une couche à l'autre. Un réseau de neurones artificiels va avoir deux principales phases : celle de **testing** qui va permettre d'obtenir une sortie en fonction de l'entrée et celle de **training** qui va permettre de modifier le poids des synapses de chaque couche pour faire correspondre correctement une sortie à une entrée. Le nombre de répétition du training est important car il permet de "forger" le réseau sur les données apprises. Un tour de training sur un jeu de données est appelée **epoch**.

L'objectif de notre démonstrateur étant la représentation de formes à partir de mouvements, nous avons dû faire un choix d'accéléromètre. Notre choix s'est arrêté sur l'accéléromètre ADXL345 qui est principalement utilisé pour des applications dynamiques. Ce capteur fonctionne grâce à un système de variation de capacité entre deux plaques, l'une étant fixe et l'autre reliée à des masses mouvantes. L'accélération dévie donc la masse et modifie la valeur du condensateur, ce qui entraîne une variation de tension en amplitude en sortie du capteur. Cette variation d'amplitude est donc proportionnelle à l'accélération et c'est celle-ci qui nous intéresse. Grâce à un convertisseur analogique numérique nous retrouvons ces données de sortie sous 10 bits qui sont transmis par I2C au microcontrôleur.

L'accéléromètre nous renvoie donc en continu une série de trois données correspondant à l'accélération selon chaque axe (x,y,z). Dans notre cas d'utilisation, nous représenterons seulement des mouvements en 2D en utilisant uniquement les données d'accélération des axes x et y (L'axe z sera dirigé vers le sol, pour s'affranchir de l'accélération due à la pesanteur).

III) Architecture générale du projet

Dans cette partie comprenant les sous-parties gestion de projet, architecture du démonstrateur réalisé et environnement de développement nous présentons dans sa totalité la manière dont nous avons organisé le projet.

III.1) Gestion de projet

Pour gérer la partie gestion de projet nous avons en tout premier lieu réalisé un lotissement de projet c'est à dire un découpage du projet en lots et sous-tâches pour lesquelles nous avons ensuite réparti les responsabilités entre les membres d'équipe (cf Annexe n°4 pour plus de détails). Nous avons obtenu le lotissement de projet suivant :

- Gestion de projet : responsable de lot Gauthier CANET ;
- Capture et représentation du Mouvement : responsable de lot Alexandre MORICEAU ;
- Reconnaissance de formes via réseau de neurones : responsable de lot Romain LE DONGE ;
- Accélération matérielle sur FPGA : responsable de lot Louka BARRIERE ;
- Intégration et tests : responsable de lot Gauthier CANET.

De là nous avons mis en place un planning à la semaine intégrant les deadlines de chaque tâche, planning mis à jour à chaque réunion interne (deux par semaine) avec les résultats d'avancement et étayé par des revues de performance (voir exemple en Annexe n°5) avant chaque réunion avec notre chargé de projet (environ une toutes les deux semaines).
Afin de faciliter nos échanges nous avons utilisé *Google Drive* pour mettre en commun tous les documents du projet (nos travaux respectifs, les comptes rendus de réunions, plannings, …).

III.2) Architecture du démonstrateur réalisé

Nous devions réaliser un démonstrateur permettant, via l'utilisation d'un réseau de neurones artificiels, de reconnaître des formes basiques à partir de mouvements puis intégrer à ce démonstrateur une accélération matérielle. Pour remplir ces objectifs nous avons mis en place le schéma fonctionnel présenté en figure 2 qui réunit les différents composants du démonstrateur à savoir l'accéléromètre, la carte de développement STM32, la carte FPGA et pour finir un PC et son écran**.** On peut également voir sur cette figure l'architecture de notre démonstrateur découpée par fonctionnalités : le bloc n°1 rassemble les éléments du démonstrateur nécessaires à la chaîne d'acquisition et de mise en forme des données. Le bloc n°2, réalisant la fonction de reconnaissance de formes sur STM32 réunit l'accéléromètre et la carte de développement STM32. Enfin, le bloc n°3 correspondant à l'accélération matérielle réunit la carte de développement STM32 et la carte FPGA.

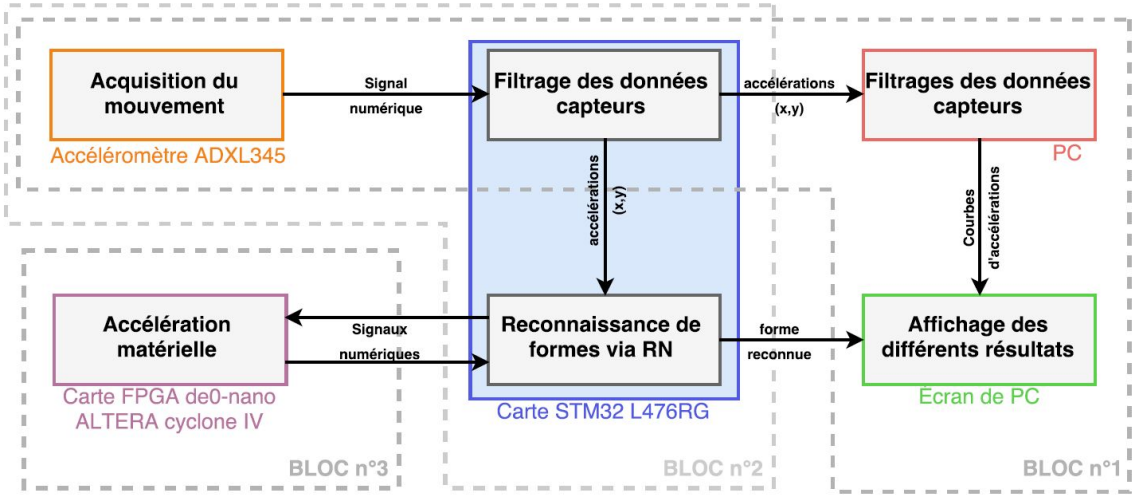


Figure 2 : Architecture globale du démonstrateur

III.3) Environnement de développement du projet

Pour mener à bien les différents travaux de ce projet nous avons utilisé plusieurs logiciels et langages de programmation. Nous présentons ici le rôle de chacun d'entre eux.

Afin d'effectuer d'une part des tests sur le ou les filtres à appliquer sur les données renvoyées par notre accéléromètre et d'autre part de représenter l'accélération correspondant aux formes que nous voulions reconnaître nous avons utilisé le logiciel *Matlab*. Les tests permettant de choisir à la fois l'architecture et les paramètres de notre réseau de neurones ont quant à eux été réalisés à l'aide du langage de programmation *Python*. Le langage *C* (avec le logiciel *System WorkBench for STM32*) a ensuite été utilisé pour implémenter sur notre carte STM32 les fonctionnalités suivantes : filtrage des données de l'accéléromètre, reconnaissance de formes via réseau de neurones et enfin interface avec la carte FPGA. Comme ces trois fonctionnalités étaient à la responsabilité de différents membres de l'équipe projet nous avons également utilisé *GitHub* pour faciliter la gestion des différentes versions de codes. Enfin, les langages de description de matériel *Verilog* et *VHDL* (avec les logiciels *Quartus* et *ModelSim* pour la simulation) ont été utilisés pour respectivement réaliser l'accélération matérielle sur FPGA et gérer l'interface avec la carte STM32 côté FPGA.

IV) Chaine d'acquisition et de mise en forme des données

Dans cette partie nous verrons quels ont été les méthodes utilisées pour récupérer les données envoyées au réseau de neurones sur STM32.

IV.1) Accélération théorique

Comme vu précédemment, l'accéléromètre ADXL345 permet de renvoyer l'accélération en temps réel suivant trois axes (x,y,z). Dans notre cas, nous nous intéressons à des figures dans un plan 2D, nous prendrons donc en compte seulement les axes x et y.

Dans l'exemple qui va suivre, nous considérerons seulement le cas de l'accélération pour le tracé d'un carré selon l'axe x.

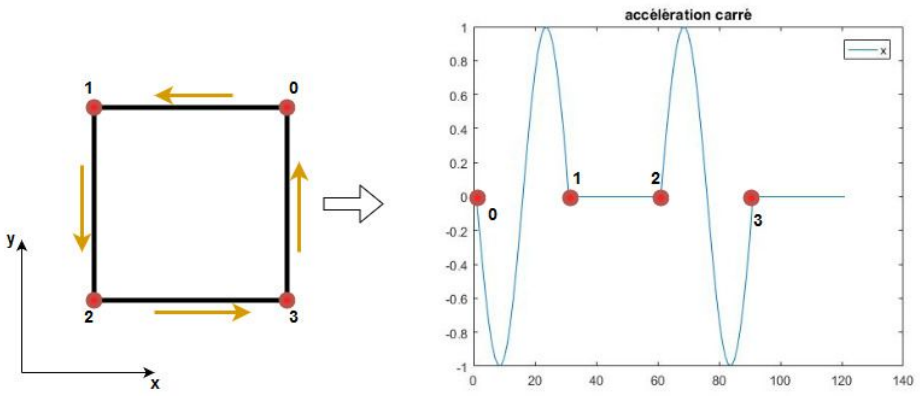


figure 3 : accélération théorique du tracé d'un carré selon l'axe x

La figure 3 ci dessus permet de bien visualiser les données théoriques qu'est censé nous renvoyer l'accéléromètre. En effet selon l'axe x, nous aurons une accélération seulement entre les points 0 et 1 ainsi que 2 et 3. L'accélération est de forme sinusoïdale puisque pour aller d'un point A à un point B il y a une phase d'accélération et une phase de décélération. Le signe positif ou négatif dépend du sens de l'accélération (et donc de l'orientation de l'accéléromètre au moment du tracé). De point de vue mathématique, le mouvement et l'accélération sont liés par une double intégrale.

IV.2) Accélération réelle et filtrage

Connaissant à présent les tracés théoriques de l'accélération nous avons cherché par le test de plusieurs filtres à s'en rapprocher au maximum. Cela nous amène aux courbes suivantes :

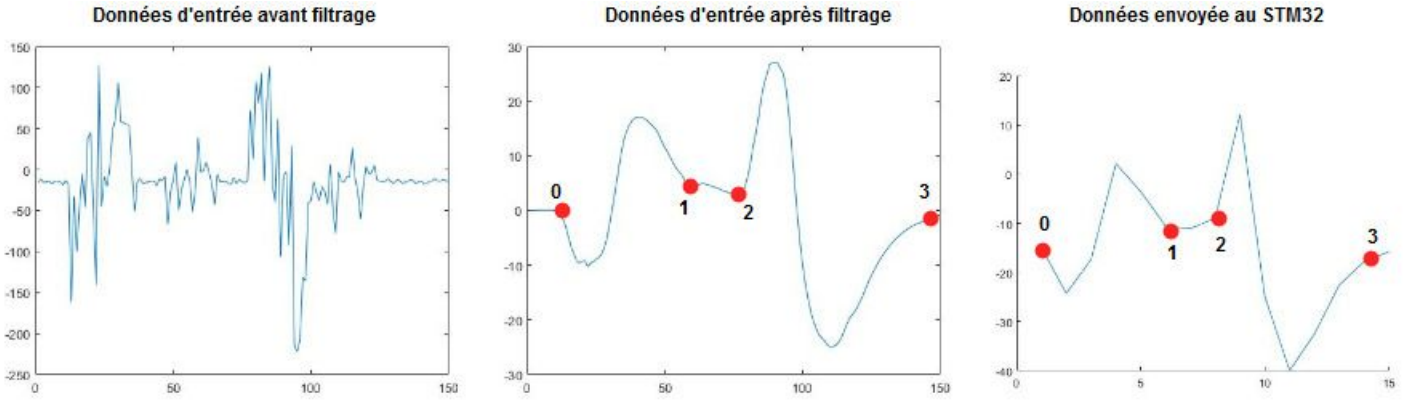


figure 4 : Chaîne d'acquisition des données

La figure 4 ci-dessus montre les différentes courbes représentant les étapes d'acquisitions des données sur un seul axe. La première courbe est la visualisation des données brutes en sorties de capteurs, cette courbe constituée de 150 points pris sur un intervalle de temps d'environ 2 secondes. On remarque la difficulté de récupérer des données à partir d'un accéléromètre en raison du bruit présent sur le signal et de la présence d'un offset. Pour éviter cela nous avons donc mis en place un filtre passe-bas pour supprimer le bruit et un filtre passe-haut pour supprimer l'offset.

Le filtre passe-bas choisi est un filtre à réponse impulsionnelle infinie d'ordre 2. En considérant $x[n]$ la suite des données brutes en sorties d'accéléromètre, la sortie filtrée $y[n]$ filtrée peut s'écrire $y[n] = 0.1 * x[n] + 0.9 * y[n-1]$. Ce filtre à l'avantage d'être facile à mettre en place et correspond parfaitement pour les signaux à temps discret que nous traitons. En ce qui concerne le filtre passe-haut nous avons utilisé un filtre à moyenne mobile. Cependant ce filtre n'est utilisé que pour l'affichage et non pour filtrer les données envoyées au réseau de neurones puisque les résultats de celui-ci ne sont pas influencés par l'offset.

Ces filtres nous ont donc permis d'obtenir la deuxième courbe de la figure 4 où l'on retrouve bien les différentes étapes de l'accélération vu précédemment sur la courbe théorique. Pour finir la troisième courbe représente les données envoyées au réseau de neurones. En effet le réseau étant adapté pour seulement 15 points par axe, nous avons donc pris seulement un point sur 10.

V) Reconnaissances de formes sur STM32

V.1) Phase de recherche

La reconnaissance de forme sur STM32 est accomplie grâce à un réseau de neurones (un perceptron multicouche) qui permet de classifier la forme effectuée à partir des données d'accélération envoyées. Nous avons pensé à intégrer deux fois ces données pour envoyer les informations de position au réseau. Mais après réflexion, pour l'algorithme du réseau de neurones cela revient au même de lui envoyer directement l'accélération. Pour créer le réseau de neurones sur STM32, nous l'avons tout d'abord implémenté en Python sur ordinateur ce qui nous a permis d'observer le comportement du réseau en fonction des données en entrées. L'utilité du script Python a été de permettre de cadrer les paramètres du réseau de neurones et plus précisément les hyper-paramètres, c'est à dire, le nombre de couches, le nombre de neurones par couches, le nombre d'entrées, le nombre d'échantillons nécessaire à l'entraînement, etc… Avec ces estimations d'hyper-paramètres nous avons codé l'algorithme de Training et de Testing en C pour pouvoir l'implémenter sur STM32. Par la suite, nous avons entraîné le réseau uniquement sur le STM32 par souci de simplicité (acquisition de données directement depuis l'accéléromètre) et pour que les calculs effectués soit le plus exact possible sur la cible finale.

V.2) Choix techniques

Nous devions effectuer une partie des calculs sur FPGA (accélération matérielle). Le FPGA ne disposant pas de FPU, il lui est plus simple d'utiliser des données en virgule fixe, c'est à dire avec un nombre de bit fixé pour la partie entière et la partie après la virgule. Nous nous sommes mis d'accord sur le format à utiliser pour effectuer les calculs sur les deux parties en analysant les poids moyens des synapses lors des entraînements précédents. Nous avons convenu que saturer les poids à [-15; 15] serait satisfaisant (ce qui nous fait 4 bits pour la partie entière) avec une précision bornée au millième (11 bits après la virgule) et un bit de signe. Nous avons donc ré-entraîné le réseau et effectué de nouvelles simulations en Python. Les résultats des tests étaient satisfaisants.

Pour définir plus précisément les hyper-paramètres, nous avons suivi les résultats obtenus avec le script sur ordinateur et nous avons fait plusieurs tests en faisant varier divers paramètres. Nous avons choisi de créer un réseau de neurones à deux couches grâce aux résultats des simulations sur Python. Nous avons prévu de reconnaître quatres formes pour le moment (cercle, carré, triangle et S) et donc d'avoir quatres résultats de sorties différents. Nous avons choisi que nos sorties seraient en "one-hot", car nous faisons ici de la classification et non

pas de la régression, cela permet de décorréler les sorties les unes des autres. Nous avons donc quatres neurones en sorties. Pour choisir le nombre d'epochs et le nombres de couches intermédiaires le choix était moins évident. En effet, Python nous a permis de dire qu'il fallait entre 30 et 70 neurones sur notres couches intermédiaires cela reste un interval vaste et le temps de training d'un réseau avec 30 ou 70 neurones varie énormément. Nous avons donc lancé plusieurs simulations sur le STM32 et tracé les résultats obtenu.

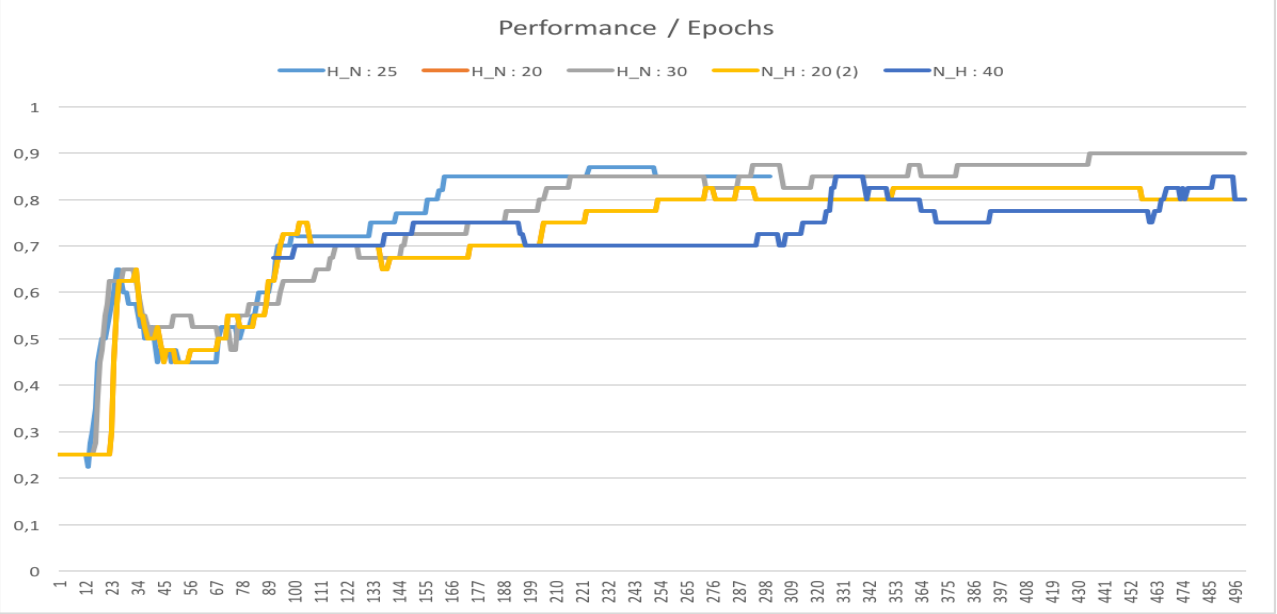


figure 5 : Evolution des performances de testing en fonction du nombres d'epochs

On voit sur le graphique précédent (figure 5) l'évolution des performances de testing en fonction du nombres d'epochs de training. Nous avons 40 données d'accélération par forme, nous avons choisi d'en utiliser 30 pour le training et 10 pour le testing. Grâce à ces courbes, nous avons pu choisir les paramètres manquant du réseau. Nous sommes partis avec 30 neurones sur la couche intermédiaire et 300 epochs pour le training.
D'un point de vue performance, quelles sont les résultats que nous avons obtenu avec un tel réseau de neurones ? C'est ce que nous allons voir dans la prochaine sous-partie.

V.3) Résultats obtenus

Avant de commencer à parler des performances il est bon de rappeler ce que nous appelons "réussite". Une "réussite" est un cas ou le réseau a réussi avec succès à identifier une forme. Notre critère de réussite est : Le chiffre le plus grand est un "1", les autres sont à "0". Ainsi dans l'exemple suivant :

{{0.004223,0.007113,0.113277,0.932655}}

Le premier nombre est considéré comme un "0", le second et le troisième également et le dernier est considéré comme un "1", ce qui nous donne {0, 0, 0, 1}.
Avec les paramètres choisis, nous obtenons un pourcentage de réussite sur les données de test allant de 85% à 90%. Le temps de testing du réseau pour une forme est de 4 millisecondes, en revanche le temps de training de ce même réseau sur STM32 est d'un peu plus de 7 minutes avec un processeur cortex-M4 à la fréquence de 48 MHz. Il y a donc un rapport d'environ 100 000 entre les deux, ce qui explique le besoin d'une accélération matérielle.

VI) Accélération matérielle sur FPGA

Pour l'accélération matérielle, nous avons décidé de réaliser le testing sur FPGA. Même si l'implémentation du training aurait été intéressante (le training étant très coûteux en temps), les délais de ce projet ne nous le permettaient pas. De plus le training n'est réalisé qu'une seule fois au préalable et n'exige pas un résultat en temps réel. Le testing, quand à lui, doit répondre le plus rapidement possible une fois le mouvement réalisé.

VI.1) Architecture du design FPGA

L'architecture interne du FPGA se décompose en 4 modules (voir figure 6):

-Main State-Machine: Gère, de manière séquentielle, toutes les étapes du testing

-Layer Memory: Mémoire RAM embarquée qui stocke les coefficients des matrices de neurones issus du training. Ils sont hardcodés lors de la programmation du FPGA via des fichiers d'initialisation de la mémoire (.mif, Intel format) et des MegaWizard Plug-In d'ALTERA.

-Calc matrix + Sigmoid + Concaténation: Réalise le calcul matriciel (colonne par colonne) avec le vecteur d'entrée et la fonction d'activation (sigmoÏde) de manière purement combinatoire. La state-machine contrôle les itérations des calculs. Le module Concaténation récupère tous les résultats de manières séquentielles et les place dans un vecteur.

-Communication Interface: Réalise le transfert des données de manière bidirectionnelle (vecteur d'entrée et vecteur de résultat) avec le STM32.

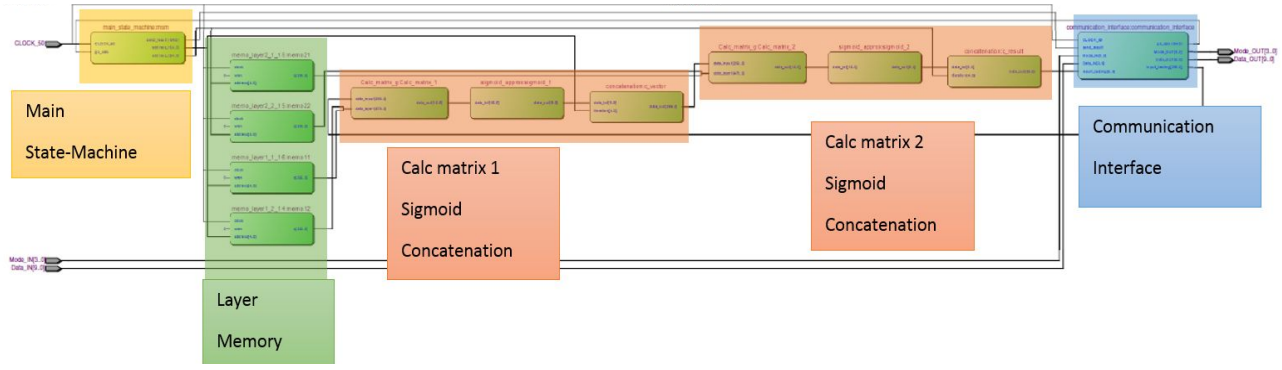


figure 6 : Architecture du design FPGA

VI.1) Tests unitaires et performances

Pour réaliser la fonction sigmoid nous avons implémenté un module approximant cette fonction avec une fonction affine par morceaux se rapprochant au mieux de la sigmoid réelle (la fonction sigmoid étant trop complexe pour être implémentée de manière mathématique). Le but étant que la précision de cette sigmoid se rapproche au mieux de celle de la cible ayant réalisé le training (STM32). Le STM32 utilise des flottants alors que nous avons choisi d'utiliser des virgules fixes codées sur 16 ou 9 bits sur FPGA pour des raisons de simplicité d'implémentation. On a déterminé que le résultat de la sigmoid est acceptable si l'erreur entre la sigmoid sur FPGA et celle sur STM32 est inférieur 0,005 pour des valeurs d'entrées allant de -15 à +15. Pour valider, nous avons réalisé une simulation de toutes les entrées possibles et comparé sur excel tous les résultats FPGA et ceux du STM32.

Nous avons réalisé ce même processus de validation par tests unitaires pour le calcul matriciel.

Pour essayer d'accélérer au plus le testing sur le FPGA, les calculs sont réalisées par des fonctions combinatoires volumineuses entraînant de multiples accès à la mémoire RAM. Nous

avons réalisé des tests directement sur la cible FPGA pour déterminer le délai nécessaire pour que les calculs soient justes . Le résultat de ce test a indiqué que le flux de données dans le chemin combinatoire nécessite quatre périodes de l'horloge disponible sur la carte FPGA (50MHz).

Le temps du testing (sans prendre en compte la partie communication qui a pour facteur limitant la vitesse du STM32) peut alors se calculer théoriquement (vérifié de manière expérimentale).

Temps Testing= (Nb de Layer* Nb d'itération * Nb de période nécessaire + 5) * période

= (2 * 30 * 4 + 5)* 1 / (50*10e6) = 5 us

On a testé la fiabilité du testing avec 40 entrées de formes différentes et utilisant les mêmes données de training que le STM32. On obtient le résultat de 36 succès (identique au STM32). Même s'il existe des légères différences dans la valeur des éléments de sorties, elles n'influent pas dans l'arbitrage du choix de la forme car elles sont minimes par rapport à la valeur des éléments.

VII) Test du démonstrateur

Dans cette partie nous présentons l'intégration de l'accélération matérielle à notre démonstrateur et les résultats des tests finaux.

VII.1) Interface FPGA/STM32

Afin d'intégrer l'accélération matérielle à notre démonstrateur nous avons dû créer un protocole de communication entre la carte STM32 et la carte FPGA puis implémenter ce protocole sur les deux cartes. L'objectif du protocole est le suivant : envoyer au FPGA les données de l'accéléromètre (après filtrage sur STM32) pour que celui-ci renvoi au STM32 le résultat du testing. La carte STM32 étant le composant principal du démonstrateur c'est elle qui tient le rôle de maître dans notre protocole de communication (et donc la carte FPGA est l'esclave). Côté STM32 on retrouve dans le code C une fonction gérant la communication et appelée dans le main et côté FPGA on trouve une machine à états codées en VHDL. Notre protocole repose sur une transmission parallèle via 28 des ports GPIO présents sur nos deux cartes : 20 sont utilisés pour le transferts des données entre les deux cartes (10 ports alloués pour l'écriture des données 10 bits côté STM32 et la lecture côté FPGA et 10 autres ports pour l'envoi en sens inverse) et 8 sont utilisés pour permettre au deux cartes de se synchroniser via des envois de requêtes et d'acquisitions (de même les ports sont divisés en 4 ports pour l'écriture côté STM32 et 4 autres pour le côté FPGA). Pour s'assurer de l'uniformité des valeurs échangées nous avons aussi ajouté à notre protocole une vérification : sur chacune des deux cartes les données reçues sont systématiquement renvoyées à l'envoyeur pour que celui-ci s'assure qu'elles sont toujours identiques.

Puisque nous voulions réaliser une accélération matérielle il nous fallait vérifier par la théorie que la communication entre les deux cartes ne constitue pas un goulot c'est à dire que le temps gagné en réalisant la phase de testing sur le FPGA ne soit pas perdu lors de la communication. En considérant un échange idéal pour l'envoi des données d'entrées au FPGA et un autre pour l'envoi des résultats du testing au STM32 et sachant que c'est la carte STM32 qui

limite la vitesse de ces échanges nous avons simulé sur *System WorkBench* l'exécution du bout de code C réalisant la communication. Le résultat de cette simulation (représenté en figure 7) nous a donné un temps d'exécution d'au moins 500 microsecondes. On constate donc avec cette simulation qu'en théorie notre communication ne constitue pas un goulot d'étranglement. En plus de cela il faut ici se rappeler que nous testons uniquement l'accélération matérielle de la phase de testing hors notre but est de démontrer que l'on peut accélérer l'ensemble du processus d'apprentissage automatique avec réseau de neurones qui comprend aussi la phase de training. Dans cette dernière les calculs effectués sont plus nombreux et plus complexes et donc susceptibles de donner un gain de temps après accélération supérieur au temps gagné pour le testing. Si nous rentrions les données de training directement dans le code du FPGA le temps de communication serait alors encore plus faible en comparaison du temps gagné par l'accélération matérielle.

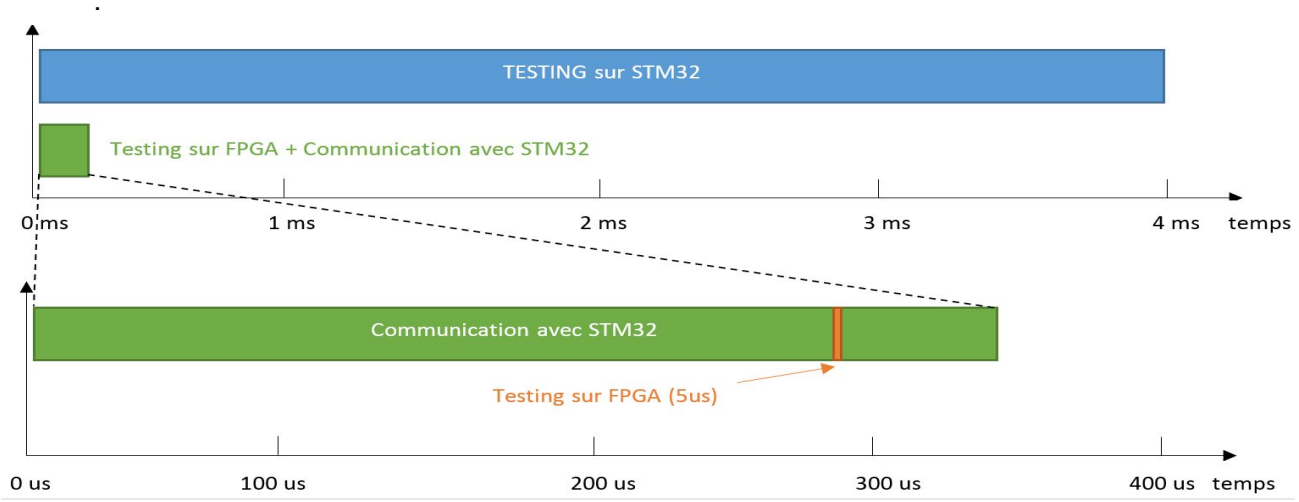


Figure 7 : Comparaison du temps de testing avec et sans accélération matérielle

VII.2) Résultats des tests du démonstrateur

Nous n'avons à ce stade pas été en mesure de finaliser l'interface de communication entre le FPGA et la carte STM32 à cause de problèmes de synchronisation que nous n'avons pas eut le temps de résoudre avant la rédaction de ce rapport. Cependant les résultats obtenus concernant le testing sur FPGA nous permettent d'affirmer que l'accélération matérielle de notre démonstrateur est réalisable. Cela prouve qu'il est possible d'accélérer de façon matérielle un réseau de neurones implanté sur système embarqué.

VIII) Conclusion

Au terme des quatres mois de ce projet d'innovation nous avons réussi à nous familiariser avec les réseaux de neurones et à en adapter un à un cas spécifique. Nous avons également réussi à réaliser un démonstrateur intégrant un réseau de neurones et capable de reconnaître un ensemble de formes basiques à partir de mouvements enregistrés via un accéléromètre et ce avec un taux de réussite de 80%. Même si notre accélération matérielle n'est pas aboutie nous avons tout de même réussi à démontrer qu'elle était possible, en réalisant sur carte FPGA la phase de testing de cette technique d'apprentissage.

Même si le temps de testing sur STM32 est bien plus élevé que celui sur FPGA, d'un point de vue applicatif, l'accélération matérielle n'apporte pas un avantage primordial en terme de performance temporelle (surtout que la communication entre les deux technologies vient diminuer le gain de temps obtenu). Nous avons donc pensé aux perspectives suivantes :

- l'association d'un stm32 et d'un FPGA n'est pas la solution technique la plus optimale en terme d'intégration. Les cibles SoC, ASIC et DSP semblent plus appropriés pour ce type de problématiques ;
- le choix d'utiliser ce type d'accélération pour le testing peut être avantageux dans des cas ou le rapport vecteur d'entrée (définit le temps de communication) sur la taille des synapses (définit le temps de calcul du testing) est faible (par comparaison notre réseau avait 1020 neurones, alors qu'un réseau de la compagnie DeepMind de Google en 2016 possédait déjà plus de 11 milliards de neurones) ;
- le training pourrait être directement implémenté sur FPGA (ou ASIC). On peut penser qu'avec le développement actuel de la robotique et des systèmes autonomes les applications futures auront sans doute besoin de réaliser leur phase de training sur système embarqué. Accélérer cette tâche aurait alors toute son utilité.

Pour finir grâce à ce projet nous avons pu découvrir le monde des réseaux de neurones. Nous avons utilisé ces algorithmes de machine learning dans un but de reconnaissance de forme. Cela nous a fait découvrir une méthodologie pour pouvoir appliquer des réseaux de neurones à un problème, comment il était possible de les intégrer sur une cible embarquée et comment grâce à des FPGA il était possible d'accélérer le temps de traitement. Toutes les compétences acquises durant ce projet vont nous permettre d'aborder les futurs projets en considérant la solution de l'intelligence artificielle.

IX) Annexes

IX.1) Résumé du projet

Pour le projet innovation nous avons choisi de nous intéresser à l'intelligence artificielle qui est de plus en plus présente dans notre environnement grâce aux objet connectés. Nous avons pour cela choisi un type d'intelligence artificielle: les réseaux de neurones.

Les réseau de neurones sont des algorithmes d'apprentissage s'inspirant fortement du fonctionnement du cerveau humain et par conséquent composés de neurones artificielles. Ils permettent par la combinaison de ces neurones d'approximer des fonctions qui s'adaptent donc à chaque situation. Le fonctionnement d'un réseau de neurones se décompose alors en deux phases, une première phase d'apprentissage qui permet de faire associer une entrée à une sortie donc de créer un modèle à partir des entrées puis une deuxième phase de test qui applique ce modèle sur des données en entrées pour "prédire" la sortie.

Ainsi pour notre cas nous avons cherché à développer une reconnaissance de formes par réseau de neurones sur une cible embarquée avec accélération matérielle sur FPGA. Pour la représentation de formes sur un plan 2D nous avons utilisé un accéléromètre connecté par I2C à un microcontrôleur STM32 sur lequel nous avons implémenté un réseau de neurones. Nous avons ensuite connecté ce STM32 à un FPGA pour réaliser une accélération matérielle et donc optimiser les performances de notre système.

A partir de cette configuration, nous avons entraîné le réseau à la reconnaissance de quatre formes différentes : le rond, le carré, le triangle, le S. Nous avons ensuite testé 40 tracés de figures et avons obtenu un résultat de 90% de figures reconnues par le réseau avec le STM32 ainsi qu'avec le FPGA dans les meilleurs cas. Le temps nécessaire à l'entraînement du réseau de neurones à été d'un peu plus de 7 minutes. Nous avons obtenu aussi un temps d'exécution du test 100 fois plus élevé sur STM32 que sur FPGA.

Ce projet nous aura donc permis de mettre en évidence l'efficacité du réseau de neurones sur système embarqué ainsi que la possibilité et l'intérêt d'une optimisation d'un réseau de neurones par accélération matérielle sur FPGA.

IX.2) English abstract of the project

Artificial intelligence is today one of the most trending topic in the field of computer science and technological advancement. This year we have chosen firstly to develop an artificial intelligence algorithm based on neural networks to accomplish motion recognition and secondly to prove that it is possible to optimize an embedded neural network by achieving an hardware acceleration of our system .

Artificial Neural Networks (ANN) are a kind of artificial intelligence algorithm that came from real biological neural networks (the same we have in our brain) and try to mimic real neurons. Following that we will talk about artificial neural networks as neural networks.

Neural networks, like the one we use in our project, need to be trained before it can get us any successful results. The training phase aims at associating an input with an output by modifying synapses weight. Once the network is trained it can performed what we call the testing phase, which simply try to predict the output with a given input. To do motion detection we have used an accelerometer which is connected to an STM32 on which we have implemented the neural network. Additionally we have connected an FPGA to the STM32 to perform the hardware acceleration and improve the performances of our project.

We have trained the neural network with many examples of four different motions : a circle, a square, a triangle and a “S”. Then we have tested the network with 40 other motions that the network was not trained on. In best cases the network succeed in predicting 39 of those new motions which means a performance of 90% on both the STM32 and the FPGA. To achieve this level of performance we needed to train the network during more than 7min on the STM32. We didn’t manage to achieve the communication between the STM32 and the FPGA but the tests we ran on both boards still gave us the following result : the FPGA is able to make the testing phase of a neural network one hundred times faster than the STM32.

With those results we have seen that neural networks accomplish good performances in embedded systems. But those performances can be improved in terms of time computation. The demonstrator we made (especially the calculation part on FPGA) is a proof of this concept.

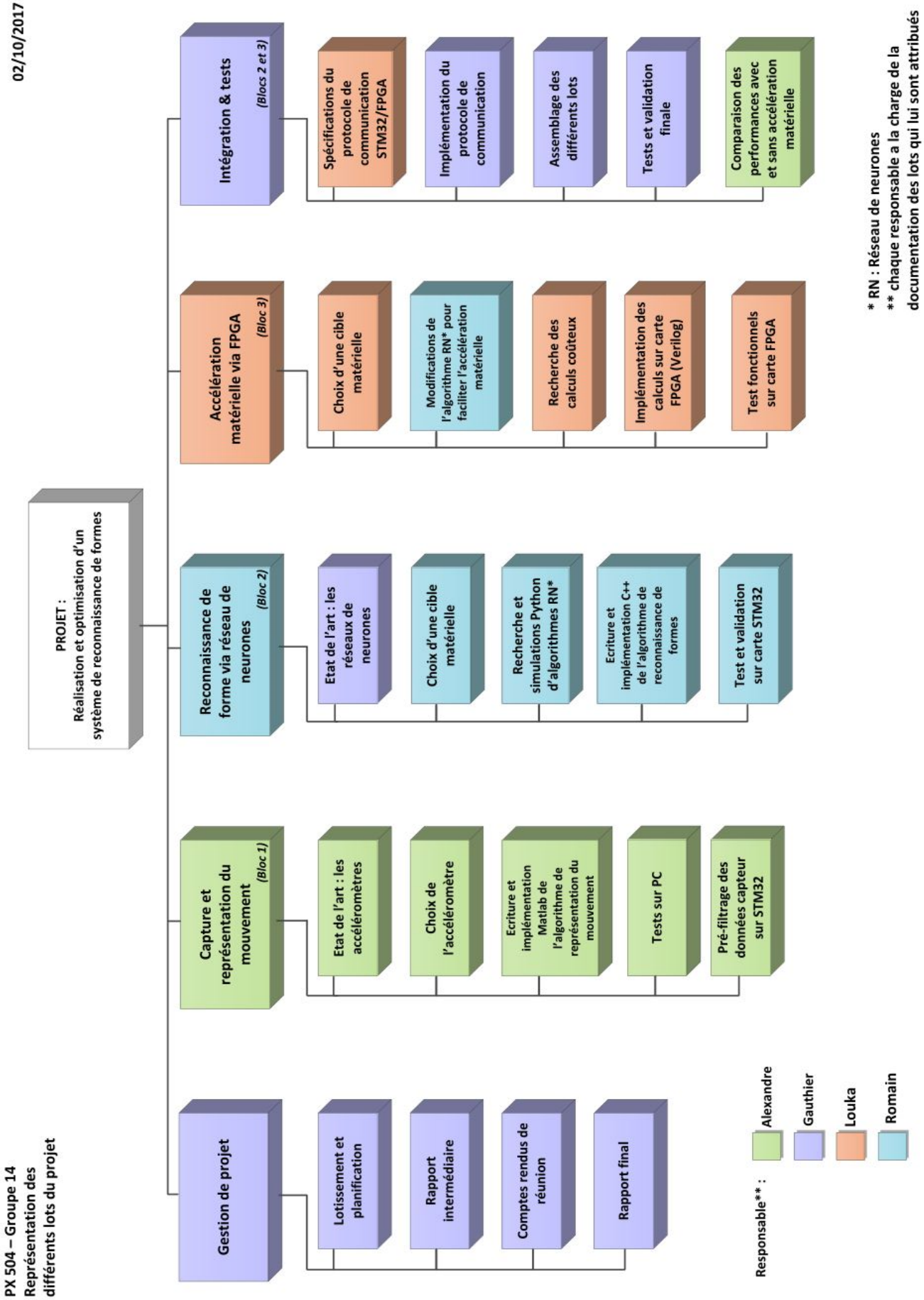
Keywords : artificial intelligence, Artificial Neural Network, motion recognition, hardware acceleration, STM32, FPGA, proof of concept

IX.3)Extrait des codes source

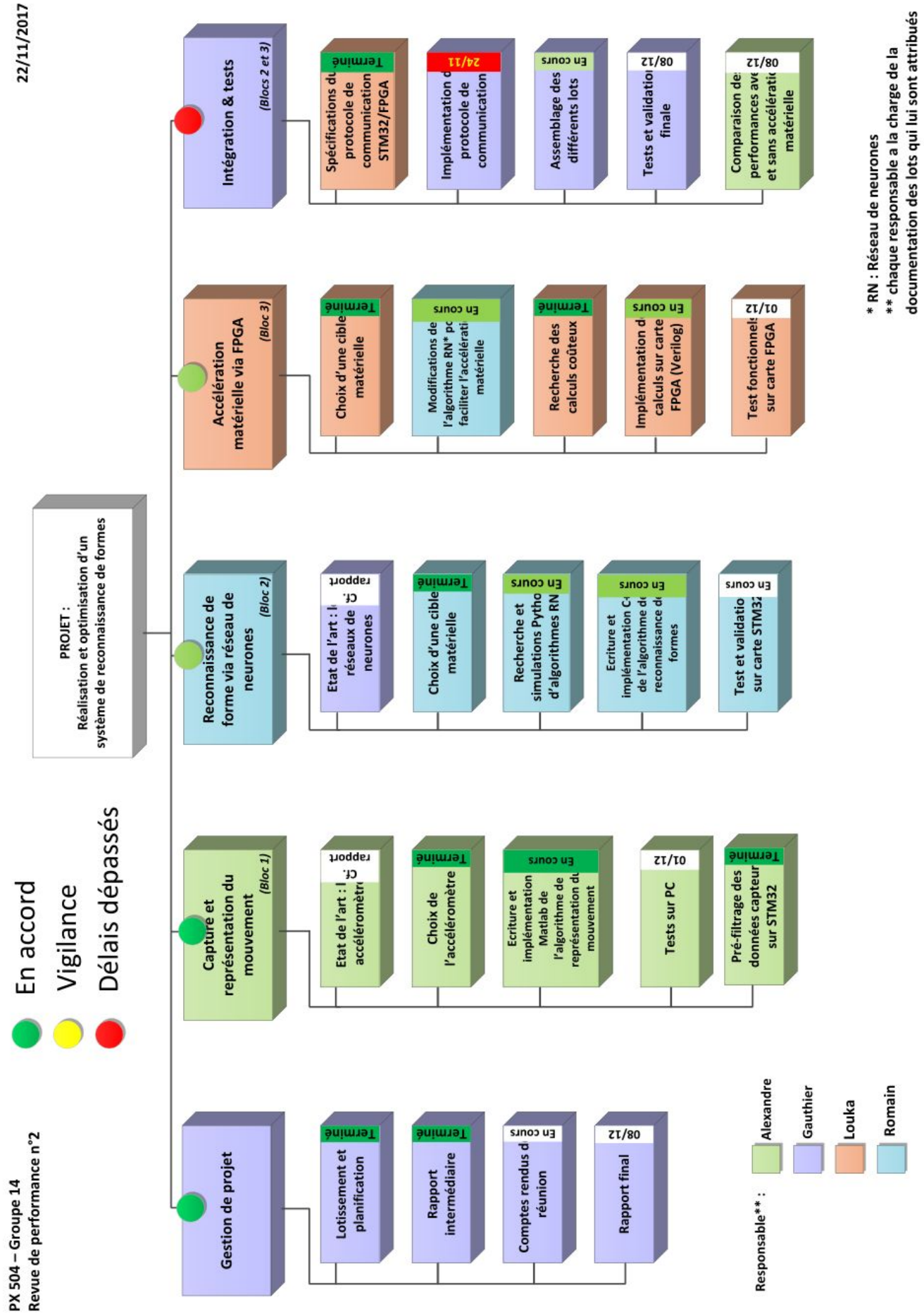
Ci-dessous l'algorithme de training utilisé par le réseau de neurones

```
int i;
for(i = 0; i < SAMPLE_SIZE;i++){//Iterate through data (inputs and output)
       //Select one value of the array
       ((VAR**)input->mat)[0] = inputs[i];
        input->matN = INPUT_SIZE;
        input->matM = 1;

        ((VAR**)output->mat)[0] = outputs[i];
        output->matN = OUTPUT_SIZE;
        output->matM = 1;

       //Testing step in the training, which is called forward propagation
        matrixProduct_float(input,syn1,l1);
        limitMatrix(l1,l1);
        sigmoid_matrix(l1,l1);
        matrixProduct_float(l1,syn2,l2);
        limitMatrix(l2,l2);
        sigmoid_matrix(l2,l2);
       //End of the forward propagation

       //Backward propagation
        matrixSubstract(output,l2,l2Error);// Calculate the delta on l2
        limitMatrix(l2Error,l2Error);
        dSigmoid_matrix(l2,l2); // Calculate the derivative of the l2
        matrixMultiply(l2Error,l2,l2Error); //Calculate the error on the l2 layer
        limitMatrix(l2Error,l2Error);
        transposeMatrix_float(syn2,syn2Transpose);
        matrixProduct_float(l2Error,syn2Transpose,l1Error);//Calculate error on l1 knowing
error on l2
        limitMatrix(l1Error,l1Error);
        dSigmoid_matrix(l1,l1Temp);//Calculate the derivative of l1
        matrixMultiply(l1Error,l1Temp,l1Error);//Calculate the error on l1
        limitMatrix(l1Error,l1Error);

        transposeMatrix_float(l1,l1Transpose);
        matrixProduct_float(l1Transpose,l2Error,syn2Temp);
        limitMatrix(syn2Temp,syn2Temp);
        matrixAdd(syn2,syn2Temp,syn2);//Modify synapses weight
        limitMatrix(syn2,syn2);

        transposeMatrix_float(input,inputTranspose);
        matrixProduct_float(inputTranspose,l1Error,syn1Temp);
        matrixAdd(syn1,syn1Temp,syn1);//Modify synapses weight
        limitMatrix(syn1,syn1);
       //end on backward propagation
}
```

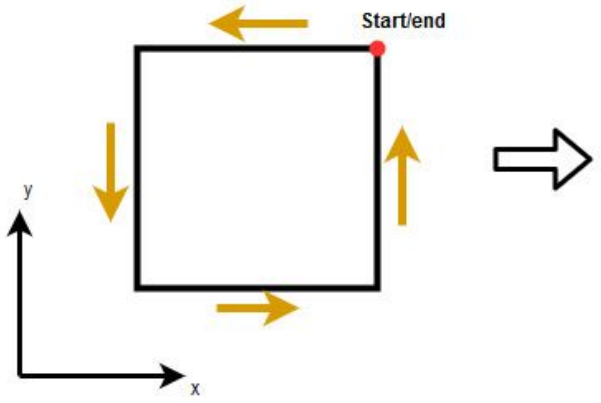
IX.4) Lotissement de projet

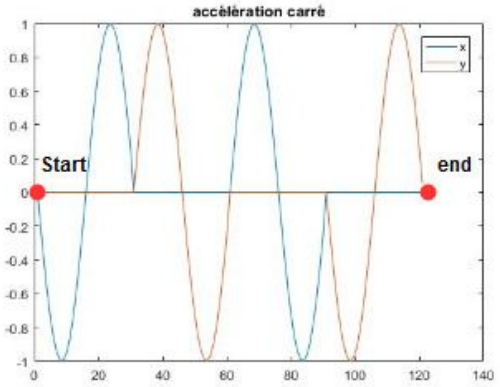


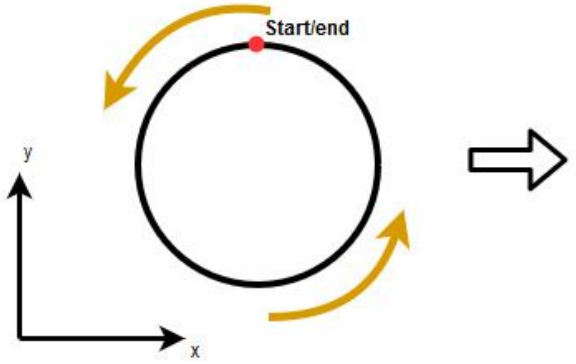
IX.5) Exemple d'une revue de performance

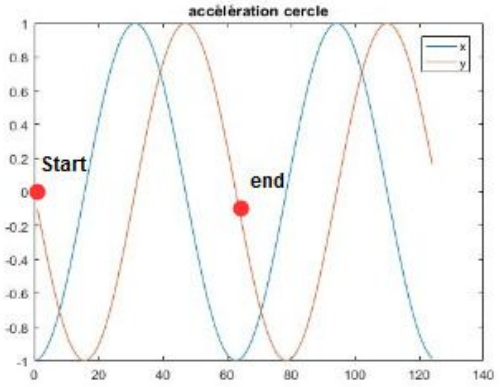


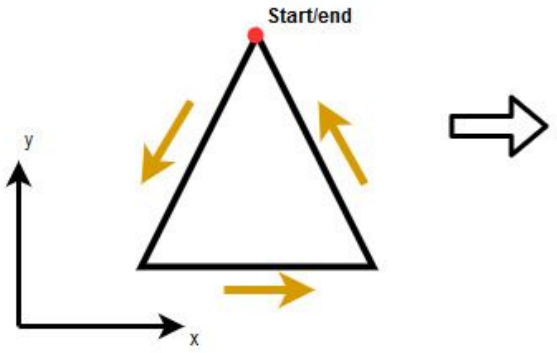
IX.6) Accélération théorique pour chaque type de figure (carré,rond,triangle,S)

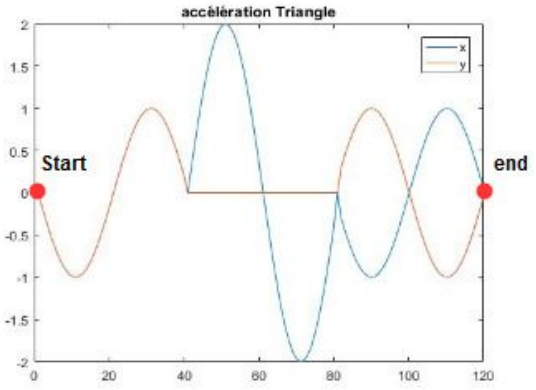


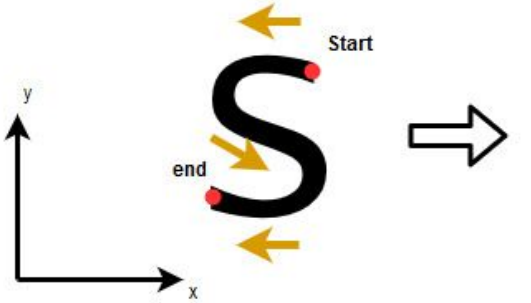


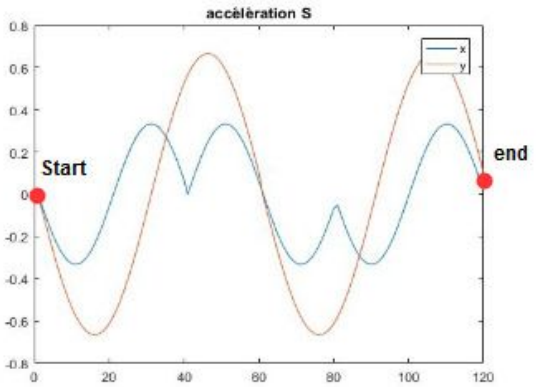






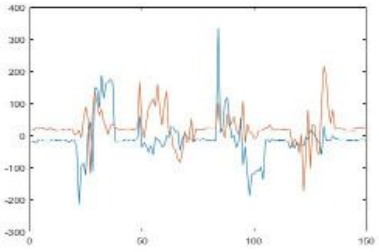




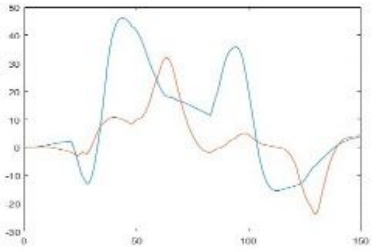


IX.7) Accélération réelles pour chaque type de figure (carré,rond,triangle,S)

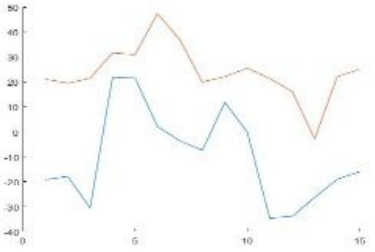
Courbes réelles pour le tracé d'un carré :



Données non filtrées

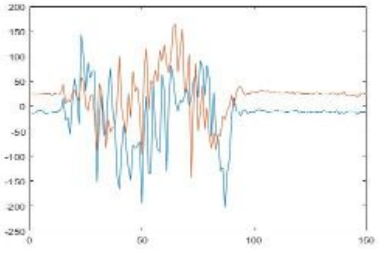


Données filtrées

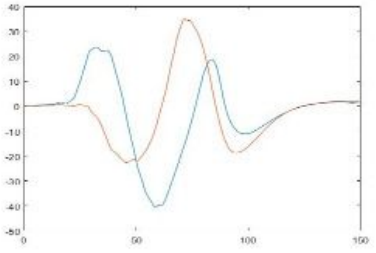


Données pour STM32

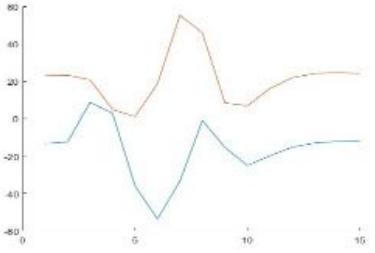
Courbes réelles pour le tracé d'un rond :



Données non filtrées

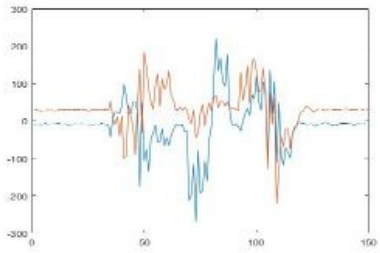


Données filtrées

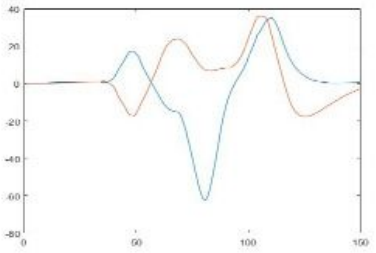


Données pour STM32

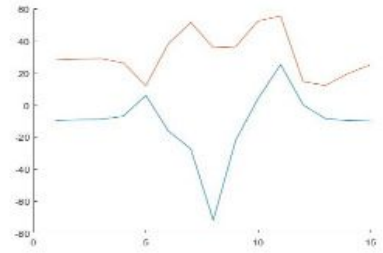
Courbes réelles pour le tracé d'un triangle :



Données non filtrées

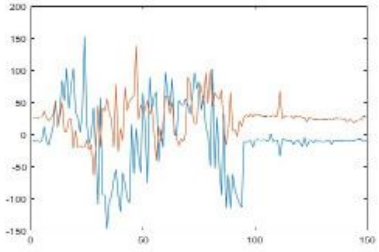


Données filtrées

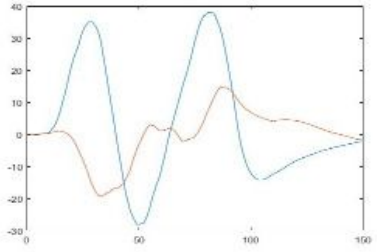


Données pour STM32

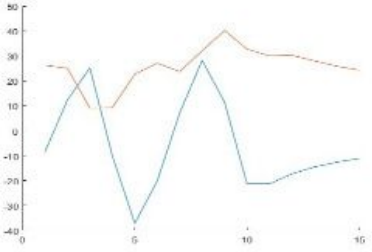
Courbes réelles pour le tracé d'un S :



Données non filtrées



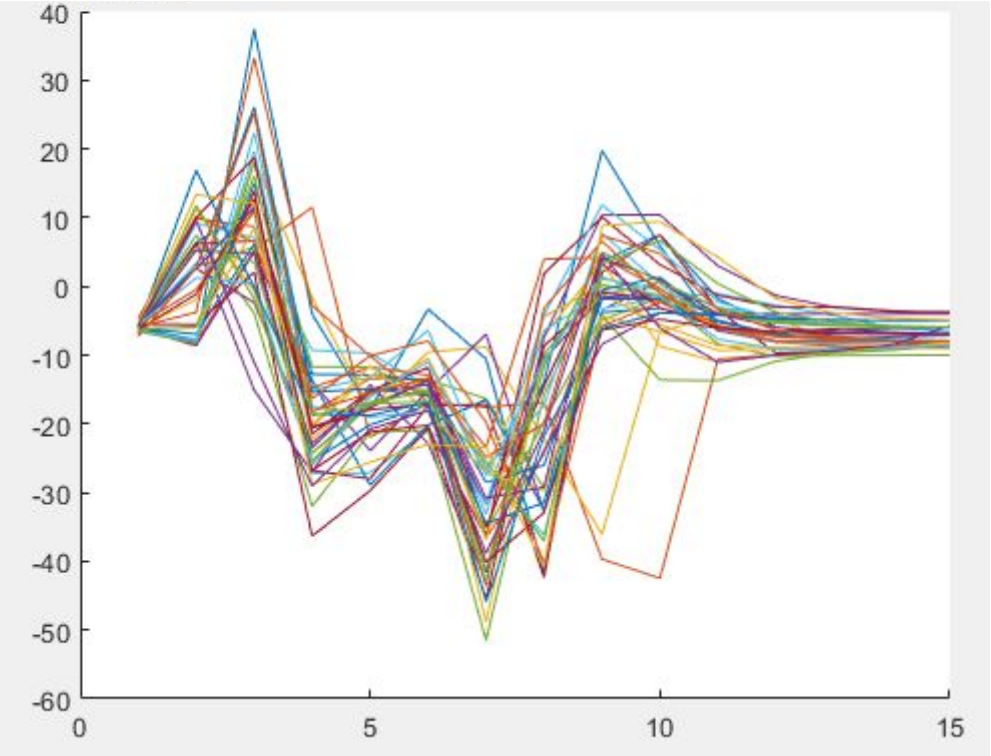
Données filtrées



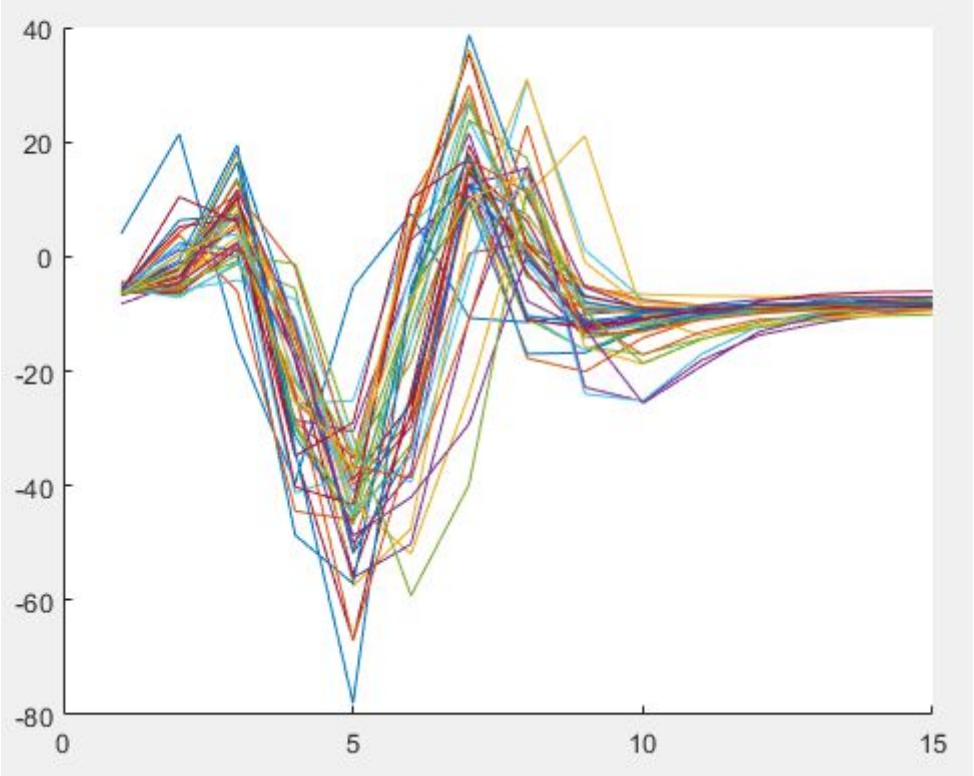
Données pour STM32

IX.8) Superposition des 40 tracés nécessaire pour le training et le test selon un axe (carré,rond,triangle,S) :

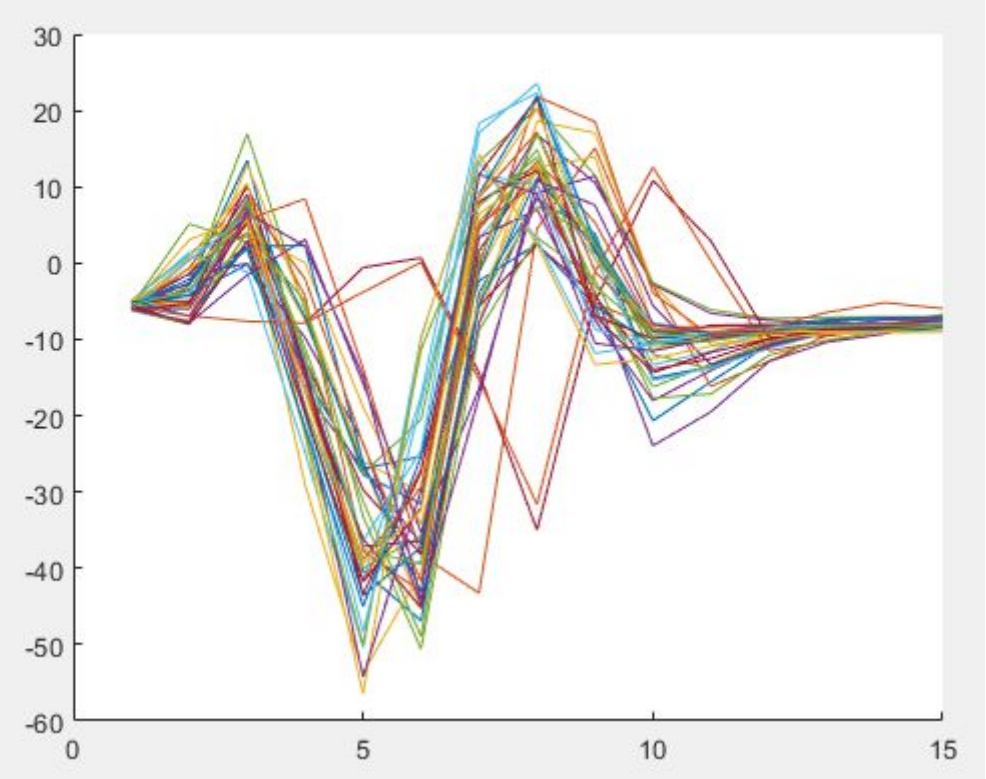
- **Dans le cas du carré :**



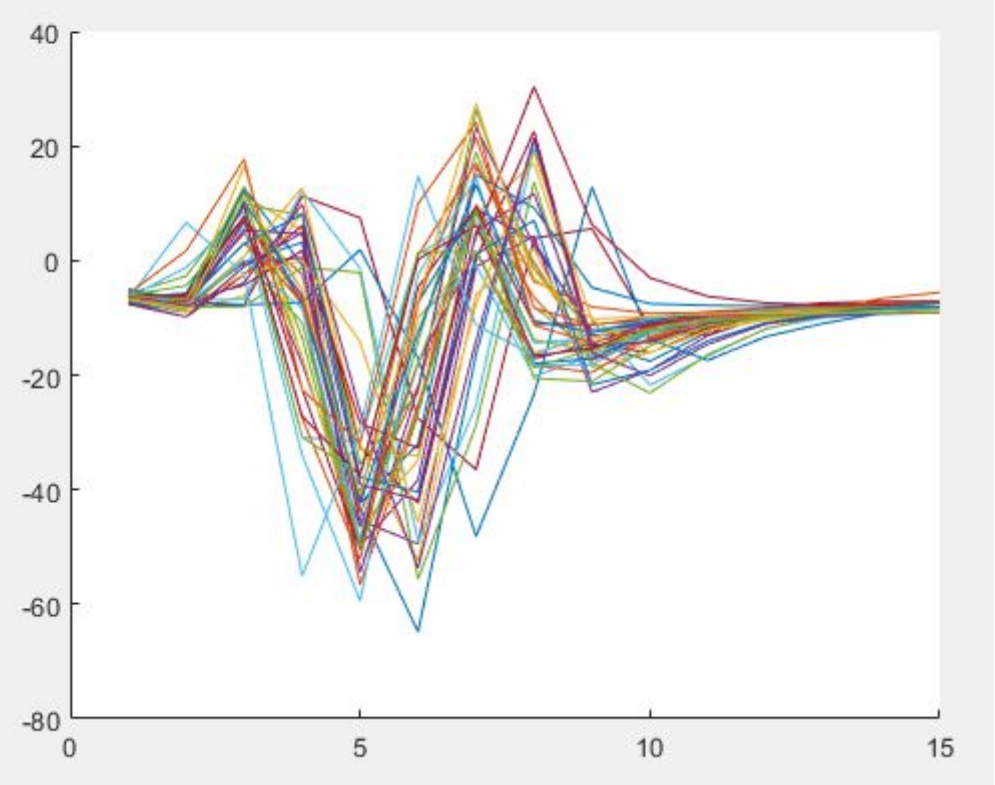
- **Dans le cas du rond :**



- **Dans le cas du triangle :**



- **Dans le cas du S :**



IX.8) Code FPGATesting top-level

```
module NN_top_level_FPGA (
input [3:0] Mode_IN,
input [9:0] Data_IN,
input CLOCK_50,
output [3:0] Mode_OUT,
output [9:0] Data_OUT
);

parameter DATA_LAYER_WIDTH = 16;
parameter DATA_INPUT_WIDTH = 10;
parameter INPUT_DIMENSION = 30;
parameter LAYER1_SIZE = 30;
parameter LAYER2_SIZE = 4;


// REGISTERS & WIRES
wire [DATA_INPUT_WIDTH*INPUT_DIMENSION-1:0]input_testing;
wire [DATA_INPUT_WIDTH*LAYER2_SIZE-1:0]result_testing;
wire go_calc;
wire send_result;
wire [DATA_LAYER_WIDTH*INPUT_DIMENSION-1:0]data_layer1;
wire [DATA_LAYER_WIDTH-1:0]data_calc_1;
wire [DATA_INPUT_WIDTH-1:0]data_sig_1;
wire [DATA_INPUT_WIDTH*LAYER1_SIZE-1:0]data_vector;
wire [DATA_LAYER_WIDTH*INPUT_DIMENSION-1:0]data_layer2;
wire [DATA_LAYER_WIDTH-1:0]data_calc_2;
wire [DATA_INPUT_WIDTH-1:0]data_sig_2;
wire [4:0]address_l1;
wire [4:0]address_l2;


// COMMUNICATION
communication_interface
#(.DATA_INPUT_WIDTH(DATA_INPUT_WIDTH),.INPUT_DIMENSION(INPUT_DIMENSION))
communication_interface(.CLOCK_50(CLOCK_50),.Mode_IN(Mode_IN),.Data_IN(Data_IN),.Mode_OUT(Mode_
OUT),.Data_OUT(Data_OUT),.input_testing(input_testing),.result_testing(result_testing),.go_calc
(go_calc),.send_result(send_result));


// MAIN STATE-MACHINE
main_state_machine #(.LAYER1_SIZE(LAYER1_SIZE),.LAYER2_SIZE(LAYER2_SIZE))
msm(.CLOCK_50(CLOCK_50),.go_calc(go_calc),.send_result(send_result),.address_l1(address_l1),.ad
dress_l2(address_l2));

// 1st LAYER CALCUL
        // CALCUL MATRIX 1
Calc_matrix_g#(.DATA_LAYER_WIDTH(DATA_LAYER_WIDTH),.DATA_INPUT_WIDTH(DATA_INPUT_WIDTH),.INPUT_D
IMENSION(INPUT_DIMENSION))
Calc_matrix_1 (.data_input(input_testing),.data_layer1(data_layer1),.data_out(data_calc_1));

        // SIGMOID 1
sigmoid_approx#(.DATA_LAYER_WIDTH(DATA_LAYER_WIDTH),.DATA_INPUT_WIDTH(DATA_INPUT_WIDTH))
sigmoid_1(.data_int(data_calc_1),.data_out(data_sig_1) );

        // CONCATENATION VECTOR
concatenation1#(.DATA_INPUT_WIDTH(DATA_INPUT_WIDTH),.NUMBER_DATA(LAYER1_SIZE))
```

```
c_vector(.data_int(data_sig_1),.data_out(data_vector),.iteration(address_l1),.CLOCK_50(CLOCK_50
));

// 2nd LAYER CALCUL

        // CALCUL MATRIX 2

Calc_matrix_g#(.DATA_LAYER_WIDTH(DATA_LAYER_WIDTH),.DATA_INPUT_WIDTH(DATA_INPUT_WIDTH),.INPUT_D
IMENSION(LAYER1_SIZE))
Calc_matrix_2(.data_input(data_vector),.data_layer1(data_layer2),.data_out(data_calc_2));


        // SIGMOID
sigmoid_approx#(.DATA_LAYER_WIDTH(DATA_LAYER_WIDTH),.DATA_INPUT_WIDTH(DATA_INPUT_WIDTH))
sigmoid_2(.data_int(data_calc_2),.data_out(data_sig_2));

        // CONCATENATION VECTOR
concatenation2#(.DATA_INPUT_WIDTH(DATA_INPUT_WIDTH),.NUMBER_DATA(LAYER2_SIZE))
c_result(.data_int(data_sig_2),.data_out(result_testing),.iteration(address_l2),.CLOCK_50(CLOCK
_50));

// RAM MEMORY BLOCK
        // LAYER 1
memo_layer1_1_16  memo11
(.address(address_l1),.clock(CLOCK_50),.wren(1'b0),.q(data_layer1[16*DATA_LAYER_WIDTH-1:0]));

memo_layer1_2_14 memo12
(.address(address_l1),.clock(CLOCK_50),.wren(1'b0),.q(data_layer1[DATA_LAYER_WIDTH*INPUT_DIMENS
ION-1 :16*DATA_LAYER_WIDTH]));

        // LAYER 2
memo_layer2_1_15 memo21
(.address(address_l2),.clock(CLOCK_50),.wren(1'b0),.q(data_layer2[15*DATA_LAYER_WIDTH-1:0]));

memo_layer2_2_15 memo22
(.address(address_l2),.clock(CLOCK_50),.wren(1'b0),.q(data_layer2[DATA_LAYER_WIDTH*INPUT_DIMENS
ION-1 :15*DATA_LAYER_WIDTH]));


endmodule
```