



ENSEA

Beyond Engineering

COMPTE-RENDU PROJET SIA

Séparation de voix embarqué sur STM32

3^{ème} année

Table des matières

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 1.1 | Objectifs | 3 |
| 1.2 | Application | 3 |
| 1.3 | Limitation ,contrainte du projet | 3 |
| 2 | Etat de l'art | 4 |
| 2.1 | Comparaison des Inference Engines pour embarqué temps réel | 4 |
| 2.2 | Le source separation and machine learning avec la méthode NMF | 5 |
| 2.3 | Le Tasnet, version LSTM | 7 |
| 3 | Software | 8 |
| 3.1 | data-generation et preprocessing | 8 |
| 3.2 | Première implémentation | 10 |
| 3.3 | Implémentation du Tasnet | 10 |
| 3.4 | Implémentations fonctionnel de Tasnet | 12 |
| 4 | Hardware | 13 |
| 4.1 | Choix de la plate-forme | 13 |
| 4.1.1 | Le Zynq 7000 soc | 13 |
| 4.1.2 | La carte STM32F746NG | 14 |
| 4.2 | Principe d'implémentation | 15 |
| 4.3 | Récupération du signal | 16 |
| 4.3.1 | Traitemen t d'interruption DMA et double buffering | 17 |
| 4.4 | Traitemen t du signal, implémentation du modèle | 19 |
| 4.4.1 | Méthode d'implémentation de notre modèle | 19 |
| 4.4.2 | Le code, création de l'application du modèle | 23 |
| 4.5 | Résultat Hardware | 25 |

1 Introduction

1.1 Objectifs

Dans le cadre de notre projet de 3ème année option SIA, nous cherchons à réaliser un séparateur de sources vocales grâce à un réseau de neurone qui sera implémenté sur une carte STM32F746G.

1.2 Application

L'idée est de pouvoir identifier la voix d'une personne dans une discussion en temps réel afin de pouvoir par exemple effectué un traitement sur cette dernière. (Traduction... Compréhension)

1.3 Limitation ,contrainte du projet

Dans notre étude nous nous limiterons au problème de séparation de voix dans un espace sans réverbérations. Le nombre de voix séparable max de notre modèle sera déterminé en fonction des performance observé lors de nos tests.

2 Etat de l'art

2.1 Comparaison des Inference Engines pour embarqué temps réel

[3] A Comparison of Deep Learning Inference Engines for Embedded Real-time Audio Classification: => article comparant les différents framework(inference engine) permettant de porter des modèles de deep learning en application audio temps réel à partir d'un modèle tensorflow (h.5) sur une Raspberry PI 4 single-board computer appairé avec un OS Elk Audio. L'article teste 4 frameworks: tensorflow-light,Torch script, ONNX runtime ainsi que RTneural sur 3 modèle de taille différentes. Les résultats laissent globalement penser que tensorlight semble

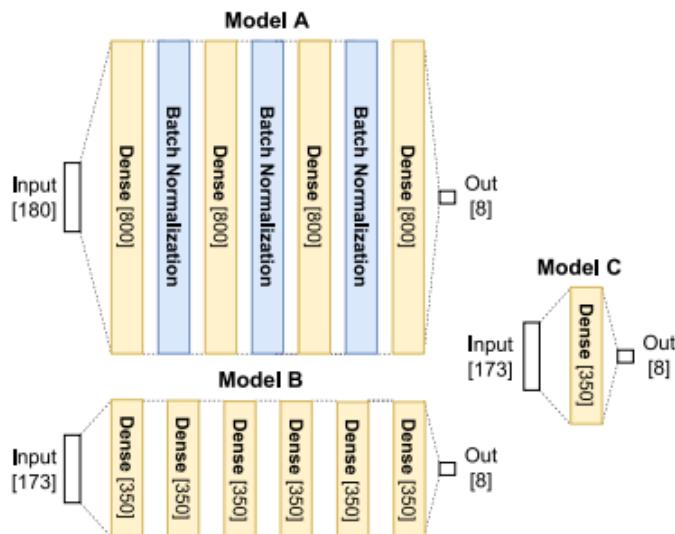


Figure 2.1: Les différents modèles testés dans l'étude

être la plate-forme la plus adaptée. En observant les temps d'exécution en "standalone" (le forward est la seul tache traité par le rapsberry) ou avec avec un plugging audio exécuté sur le Olk audio os en temps réel. On retient globalement de cette étude que le modèle à exécuter sur notre carte embarquée doit être en taille de l'ordre du modèle B ou C (avec ici un temps d'exécution max de l'ordre de 10 ms avec la limite fixé à 30 ms éviter la sensation de latence). Ce graphique indique quand à lui qu'il est qualitativement plus facile d'utiliser d'une façon général TFflite, dans la suite on a donc essayé de développer des modèles h5 via tensorflow.

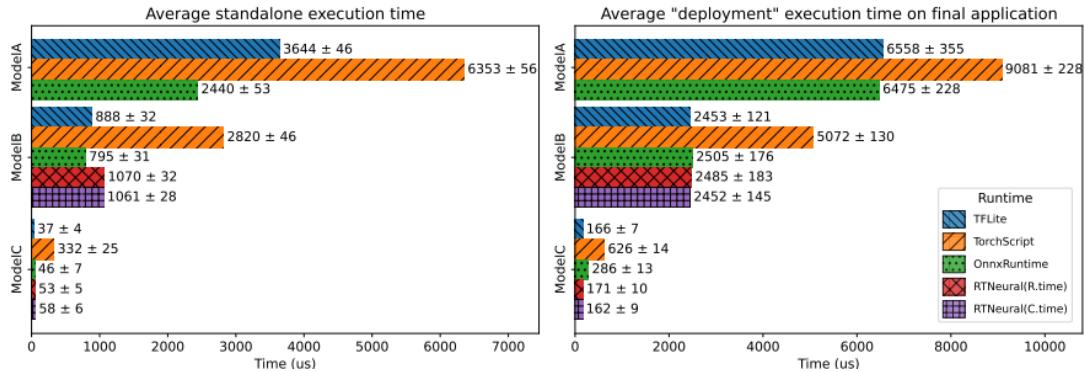


Figure 2.2: Temps d'exécution du forward

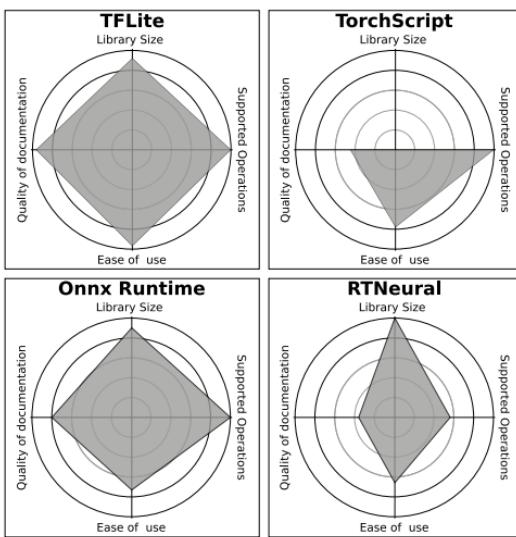


Figure 2.3: Les qualité de chacun des IE

2.2 Le source separation and machine learning avec la méthode NMF

[4] Le "Source separation and machine learning" de 2018 par Jen-Tzung Chien dresse un état de l'art général des techniques jusqu'en 2018 utilisées pour la séparation de source. Il y apparaît que le problème se pose en terme de nombre de sources à estimer et en nombres de canaux de détections du signal. Vu les contraintes de notre projet, on ne s'intéresse qu'au contenu traitant de la séparation mono-canal (on n'utilise qu'une carte) multi-sources aveugle (on veut distinguer plusieurs voix sans en connaître le nombre préalable), sans écho/réverbération (à priori dans des salles très absorbante acoustiquement parlant ou espace ouvert). Parmi les techniques abordées, on peut compter la NMF (Non negative matrix factorisation) qui consiste à approximer le signal par un produit de matrice positive $X = B.W$ avec B une base

estimé par l'entraînement et W une matrice de poids .L'idée est que les vecteurs Bases sont les mêmes pour tous les X et constituent donc un genre de décomposition du set d'entraînement à partir duquel ils sont calculés. les dimensions pour avoir K bases de matrice sont $X(N, M) = B(N, K) * W(K, M)$. Dans le cas de l'audio, on peut prendre en vecteur X le module du spectrogramme du signal audio qui est une matrice de taille N, M de coefficient positif. L'entraînement supervisé à lieu comme suit.

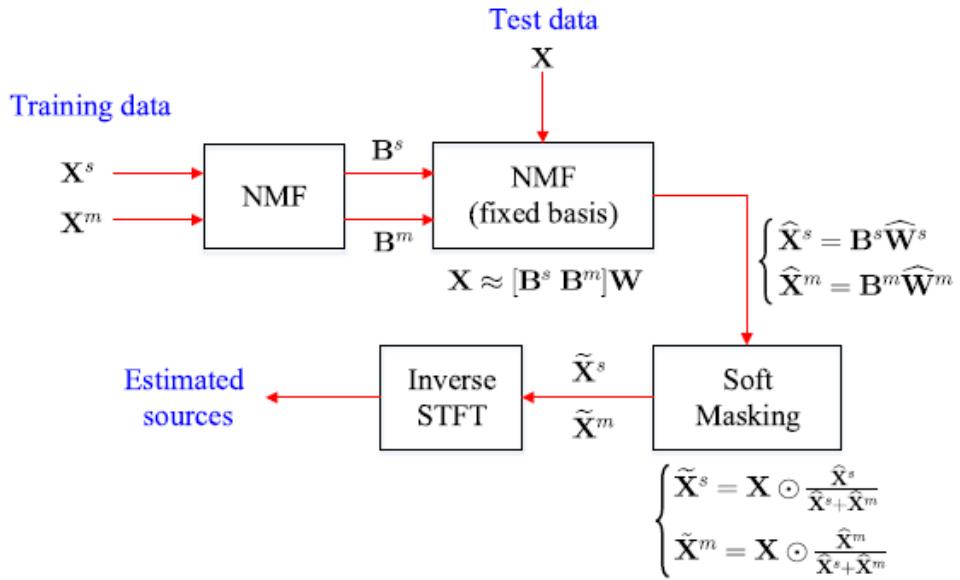


Figure 2.4: Apprentissage supervisé de la NMF (provient du livre source separation)

Cette méthode a inspiré l'algorithme que le modèle que l'on a choisi utilisera dans l'implémentation la plus avancée de notre projet: Le Tasnet.

2.3 Le Tasnet, version LSTM

[5] Nous détaillerons le fonctionnement du Tasnet dans la suite puisque nous l'avons implémenter. Nous mentionnerons simplement ici que le Tasnet suit un principe analogue à la NMF puisque l'encodage calcul ici K vecteurs Basis(positif par relu*sigmoïde toute deux positive) et que le réseau cherche alors les vecteurs basis à trouver pour reconstituer les signaux "d'entrées". Il a cependant l'avantage de ne pas nécessiter une transformation en spectrogramme du signal. Nous avons choisi cette implémentation car elle a été pensée pour

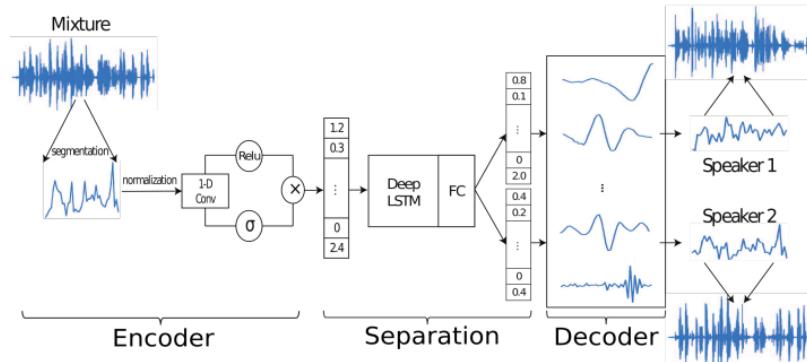


Figure 2.5: Le modèle Tasnet

le temps réel en embarqué (elle est d'ailleurs en terme de taille de même ordre que celle vu dans l'article 1). On peut aussi le voir comme un auto encoder, qui réduit à l'extrait sonore à ses information essentiel pour ensuite en étudier les corrélations temporels et en déduire par apprentissage supervisé l'appartenance à la voix 1 ou 2. Par ailleurs cette article définit et utilise les métriques classique pour évaluer la performance de la séparation: le si-SNR(scale invariant signal to noise ratio) qui est comme son nom l'indique une version normalisé du SNR où le signal utilisé en référence est le signal s_1 par exemple et le bruit défini comme la différence entre l'estimation de s_1 et s_1 . Ci-dessous la définition du si-snr. et le sdr(signal

$$\mathbf{s}_{target} = \frac{\langle \hat{\mathbf{s}}, \mathbf{s} \rangle \mathbf{s}}{\|\mathbf{s}\|^2}$$

$$\mathbf{e}_{noise} = \hat{\mathbf{s}} - \mathbf{s}_{target}$$

$$SI-SNR := 10 \log_{10} \frac{\|\mathbf{s}_{target}\|^2}{\|\mathbf{e}_{noise}\|^2}$$

Figure 2.6: Les différents modèles testés dans l'étude

to distortion ratio) mesure analogue si ce n'est que le bruit est corrélé au signal. La mesure semble cependant plus contesté. [6]

3 Software

3.1 data-generation et preprocessing

Pour venir à bout de ce projet, la première étape à était de trouver des données, en l'occurrence des enregistrement de voix. Cherchant des voix en français que nous pourrions facilement manipuler. Nous avons téléchargé 30 extraits de livre audio sur le site litterature-audio. Nous les avons alors converti au format wav, ré-échantillonner à 11025 Hz(à la base ils sont échantillonné à 44100 Hz) et pris 3 extract par livres de 5 secondes. De là, on applique divers traitement puis on somme linéairement deux signaux deux à deux en sélectionnant aléatoirement parmis les $96 \times 7 = 672$ traitements différents. Ces opérations sont résumés sur le documents ci-dessous. On ajoute alors éventuellement un bruit blanc à environ 2 tiers des signaux. les paramètres des effets sont:

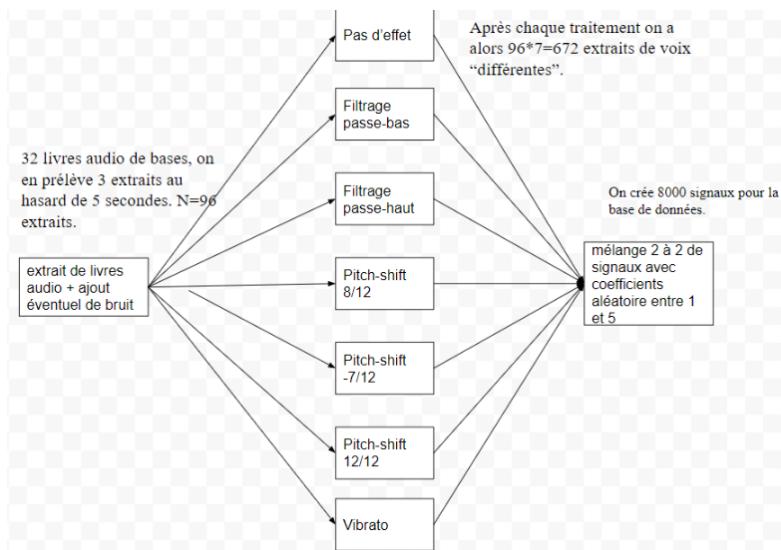


Figure 3.1: Schéma de la créations des données sonores

1. $fc=1000$ Hz, butterworth passe-bas
2. $fc=700$ HZ, butterworth passe-haut
3. pitch-shift= $8/12$ (fréquence multiplié par $2^{\frac{8}{12}}$ pour chaque composante)
4. pitch-shift= $-7/12$

5. pitch-shift=12/12
6. Vibrato A= 300 de fréquence f=5hz.

le vibrato est ici une modulation en fréquence sinusoïdale du signal, il est défini par le code suivant: avec un A de 300, le retard maximal induit par cette modulation est de

```
def Vibrato(entree,f_ech,f,A):
    out=np.empty_like(entree)
    for i in range(len(entree)):
        m=round(A*((1+np.cos(2*np.pi*(f/f_ech)*i))/2))
        if i<=m:
            out[i]=entree[i]
        else:
            out[i]=entree[i-m]
    return(out)
```

Figure 3.2: Code du vibrato(modulation de fréquence)

$\frac{300}{44100} \simeq 0.006s$. On génère ainsi un data-set de 8000 samples(5000 pour le train, 2000 pour le validation et 1000 pour le test set) pour 5*11025 features(les échantillons) sous format csv. On génère également le data-set info, contenant pour chaque samples ses informations(fréquence d'échantillonage, nombre de voix le composant, effet sur les voix , bruit etc.) là encore sous format csv.Pour pallier aux éventuels doublons,

(en effet $p(\text{aucune combinaison}) = \frac{\binom{674}{2}!}{\binom{674}{2}^{8000}((\binom{674}{2}-8000)!)}$) presque nulle.

Il y a donc nécessairement quelque doublons. On peut néanmoins estimer que par la multiplication par un coef aléatoire puis re-normalisation, on n'a que des signaux différents.

Le code python correspondant est dans le fichier extract-data-construct.py. Le code create-data.py fait sensiblement la même chose mais ne crée pas de dataset, il écrit seulement dans des dossier les fichier wav mix avec les s1 et s2 leurs correspondant dans des dossier. Ils servent sous cette forme pour l'entraînement et le test du Tasnet que nous verrons en fin de parti Software.

3.2 Première implémentation

Une fois les dataframes sous format csv générés, on a cherché à implémenter l'architecture suivante de modèle. On a donc commencé à entraîner des modèles CNN pour déterminer

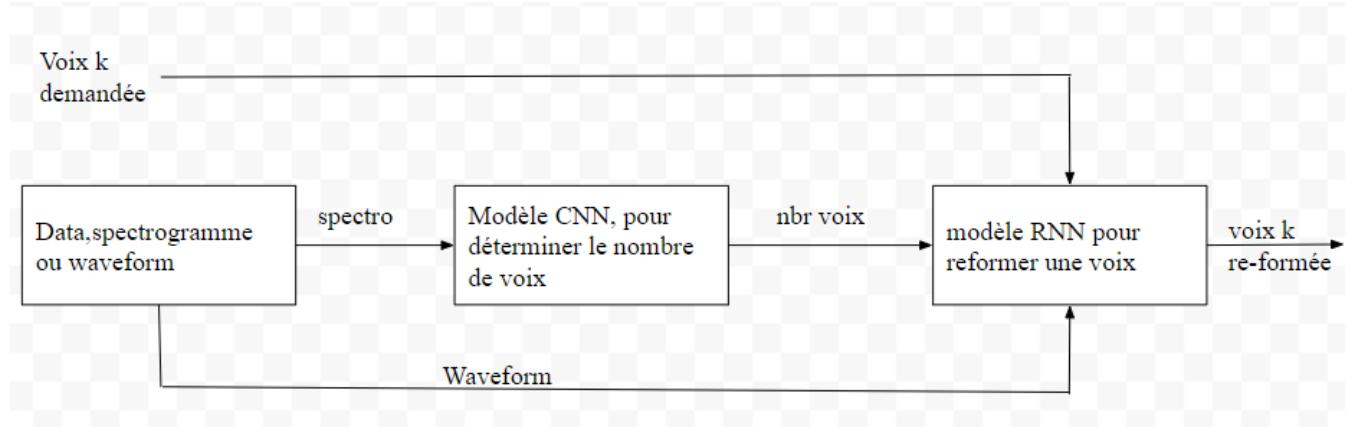


Figure 3.3: Modèle de notre première idée d'implémentation

le nombre de voix dans le signal, c'est donc un problème de classification. Les résultats étant peu probants pour séparer 10 ou 5 voix, on décida d'abord de tenter une séparation de 1 à deux voix. Là encore, les résultats sont peu décisifs. (résultat dans le notebook modèle-Cnn-sia.ipynb). Globalement, le modèle semble incapable de distinguer efficacement les différentes configurations. Dans la suite on s'est donc tourné vers une implémentation ayant fait ses preuves: le Tasnet.

3.3 Implémentation du Tasnet

Nous avons repris et adapté l'implémentation en python du tasnet de ce répertoire:github-tasnet, il a fallu alors adapter notre dataset à ce projet d'où le code `create-data.py` et effectuer les changements liés à l'update de pytorch de 0.4 à 1.4 (c'est en effet une implémentation via pytorch). Le code s'articule de la façon suivante:

1. Le code `preprocess-12.py` utilise les données correctement rangées pour créer des fichiers json contenant la taille et l'emplacement des extraits audio constituant le dataset.
2. `train.py` permet l'entraînement et l'instanciation du modèle et des données en faisant appel au `solver.py`
3. `separate.py` permet d'effectuer la séparation sur des fichiers audio en entrée en donnant le chemin du modèle précédemment entraîné.
4. `evaluate.py` permet quand à elle d'évaluer les performances sur les données de test (appelées validation dans notre projet) en calculant le si-snir et le SDR moyen des données de test.

le code python correspondant à cette partie se trouve dans le dossier Tasnet. Concernant son fonctionnement et principe: le Tasnet prend en entré un tensor de Taille $B*k*l$ où B est la batch-size, K le nombre d'extrait de taille l dans un sample et l la longueur d'un de ces extraits(dans notre cas on a des extract de 5 ms à 11025 Hz soit $l=55$). Pour obtenir un entraînement plus réaliste, la taille k des extract est aléatoirement choisi entre 1s et 4s dans le code `create-data.py`, le projet choisi alors pour traité les donné comme K le k max parmi tout les samples du batch. De la le réseau calcule les vecteurs B et W de la méthode NMF via l'encoder et le separator respectivement un réseau CNN et un réseau LSTM(voir figure 2.5). l'idée est ensuite de reconstituer à partir des X_1 et X_2 estimé en sortie du separator de reconstituer le s_1 et le s_2 , c'est le rôle du décodeur qui est simplement un réseau dense de dimension appropriée. On obtient alors les résultat suivant en utilisant la méthode evaluate sur nos données de test. Les résultat étant sensiblement moins bon que dans l'étude original(de

Average SDR improvement: 2.57
Average SI-SNR improvement: 2.33

Figure 3.4: SI-SNR,SDR moyen calculé sur le testset

l'ordre de 8 dB), on estime qu'ils peuvent s'expliquer par le manque de variété de nos données et par l'utilisation de l'effet de vibrato qui induit peut être des données trop excentrique pour le modèle. En dernière étape de cette partie software, une fois un bon modèle entraîné, nous avons créer un programme pour enregistrer et séparer deux voix via les périphériques audio d'un ordinateur

3.4 Implémentations fonctionnel de Tasnet

Cette implémentation est dans le code record and separate, qui comme son nom l'indique record un audio dans le format requis(wav ,fs=11025hz,format 16 bit int) pour une durée que l'on peut choisir (mais théoriquement supérieur à 5ms). l'idée est donc simplement de record un extrait de N2 seconde,(en exemple dans le code 10). On ouvre un stream pyaudio qui capte buffer par buffer (de contenance 1024) l'audio, le ferme et le sauvegarde. On applique alors à cet audio la fonction preprocess de preprocess-12.py et enfin la fonction separate sur le fichier audio accompagné du json créé précédemment. La procédure du code record et separate est résumé dans le graphique ci-dessous. Grâce à cette implémentation test, nous sommes assez

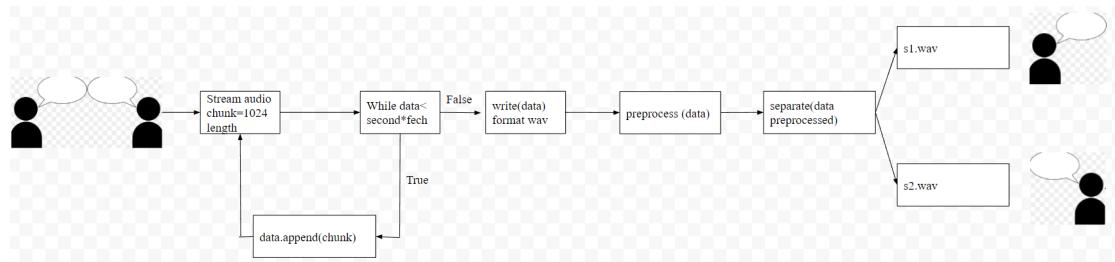


Figure 3.5: Schéma d'implémentation de record and separate

confiant quand à la portabilité théorique de ce modèle car le temps de forward semble tout à fait négligeable dans cette implémentation. Reste alors à l'importer en embarqué.

4 Hardware

Après implémentations de notre projet sur un ordinateur en langage python, nous cherchons désormais à implémenter notre modèle sur une carte. L'objectif est de permettre d'effectuer de la séparation de source en temps réel.

4.1 Choix de la plate-forme

Lors de ce projet, nous avions le choix entre une multitude de plate-forme de développement.

4.1.1 Le Zynq 7000 soc

[1]

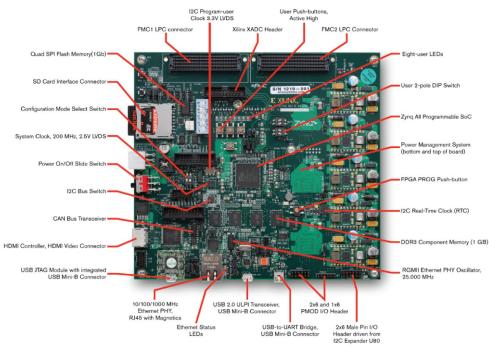


Figure 4.1: Zynq702

Pendant un temps, nous pensions implémenter notre séparateur sur une carte ZynQ 7000 soc. Cette plate-forme possède de nombreuses qualités. En effet, le ZYNQ est composé d'une partie PS (software programmable) et d'une partie PL (logique programmable). La PS comprend deux processeurs ARM Cortex-A9, la partie PL contient un FPGA. Pour faire le lien entre la PL et la PS, il y a un canal AXI.

Un peu plus en détail :

1. La partie PL (FPGA). Cette dernière permet de traiter les signaux avec de très hautes performances grâce à la parallélisation des calculs et à une fréquence de fonctionnement élevée de 50 MHz. Elle est également rétro-programmable, ce qui lui confère une grande flexibilité.
2. La Ps (le processeur double cœur ARM Cortex-A9) est le contrôleur de la carte. Il gère l'exécution du code principal et l'initialisation de la PL. La carte fournit également

1GB de RAM DDR (Double Data Rate) qui est la mémoire du système. Les transferts DDR ont de très grandes performances car la DDR transmet les données sur les fronts montants et descendants.

Cette carte malgré ses qualités nombreuses : (performance , capacité de parallélisation , mémoire importante) est compliquée à programmer. De-plus elle n'était pas disponible rapidement pour notre projet et les outils d'implémentations sont réputés complexes. Ainsi étant limité par le temps nous avons décidé de travailler sur la carte STM32F746NG qui était disponible tout de suite.

4.1.2 La carte STM32F746NG

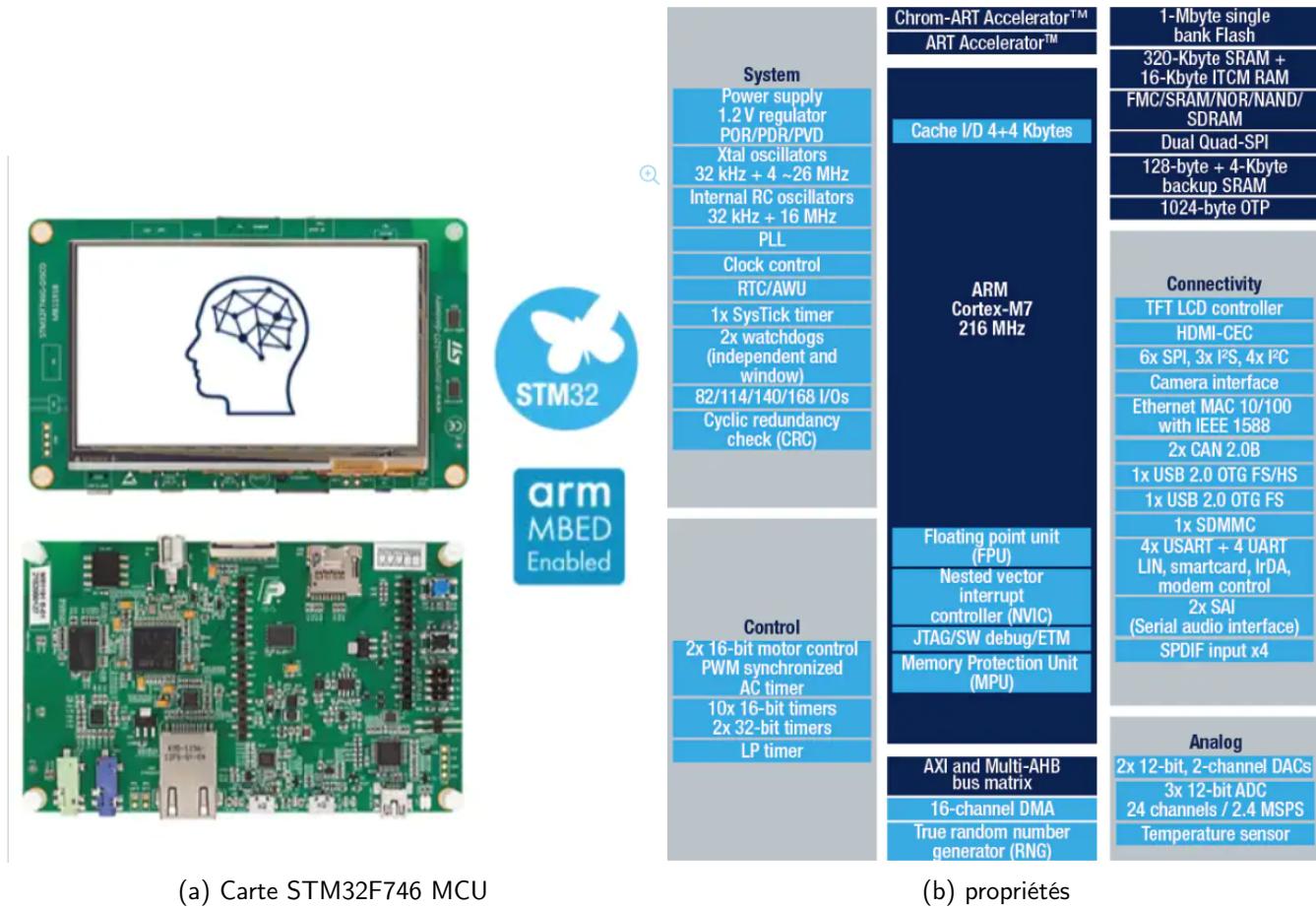
[2]

Cette carte possède de nombreux avantages: Elle est peu chère, facile à programmer via l'ide STM32 cube Mx et surtout elle est compatible avec le software pack X-cube-AI. Une librairie qui nous permettra de tester et d'implémenter facilement notre modèle dans la carte.

Cette carte possède également deux périphériques audio : composés de deux micros et d'une sortie ca10 out qui nous permet de brancher un casque audio.

Cependant ces capacités de stockage sont faibles. Elle ne dispose que de 1 Moctects de mémoire flash ,320 Koctets de SRAM (dont 64 Koctets de Data TCM RAM accessible), 16 Koctets d'instruction TCM RAM (pour les routines en temps réel). Cette faible capacité de mémoire aura un impacte sur l'implémentassions de notre modèle.

De plus elle ne dispose pas de capacité de parallélisation, cependant elle compense cela par l'utilisation du noyau RISC 32 bits ARM Cortex-M7 fonctionnant à une fréquence de 216 MHz. Le cœur Cortex-M7 est doté d'une unité à virgule flottante unique (SFPU) qui augmente considérablement les performances de calcule (addition,multiplication) entre float. Utile lors de l'exécution de notre modèle.



(a) Carte STM32F746 MCU

(b) propriétés

Figure 4.2: STM32F746NG MCU Property

4.2 Principe d'implémentation

L'idée pour implémenter notre projet est de séparer notre implémentation en quatre tâches distinctes, La partie view, modèle, périphérique et application.

1. View: Cette partie consiste en la gestion de l'interface utilisateur, la gestion des boutons.
2. Application: Cette partie fait le lien entre les autres modules. C'est le code qui exécute notre séparateur vocale. il traite le signal audio d'entrée et renvoie les sources séparées.
3. Audio processing: Ce module manage les périphériques d'entrées et de sortie audio, enregistre dans la mémoire le signal audio à l'aide d'un système de double buffering.
4. Modèle : Le modèle est le module qui manage la mémoire de notre carte. Il alloue de l'espace pour enregistrer les poids du modèle IA, ainsi que les signaux audio.

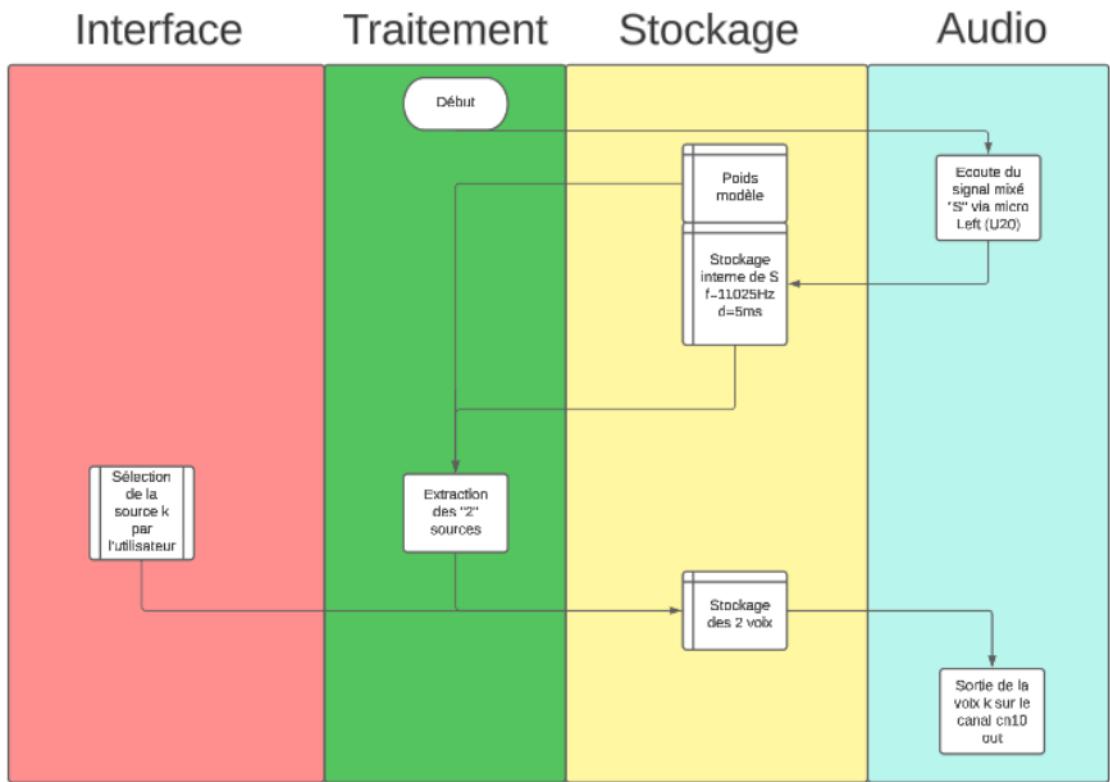


Figure 4.3: Schéma l'implémentation sur la carte

4.3 Récupération du signal

Pour cette partie nous nous inspirons très largement du TP audio SIA que nous avons modifié pour s'adapter au cahier des charges de notre projet. En d'autre termes, nous avons supprimé la partie affichage graphique qui n'est pas nécessaire dans notre projet. De plus l'utilisation des ressources graphiques nécessite un espace mémoire très important. Or notre modèle nécessite déjà beaucoup d'espace mémoire pour tourner. Nous réutilisons donc la partie sur les interruptions DMA

Principes clés de notre Implémentassions

1. Le "double-buffering" est une technique d'optimisation informatique qui permet d'augmenter le débit input output. En effet l'idée est de diviser le buffer en deux. Ainsi lorsque l'on transmet des données dans la première moitié du buffer le processeur peut travailler sur la seconde et inversement. voir 4.4

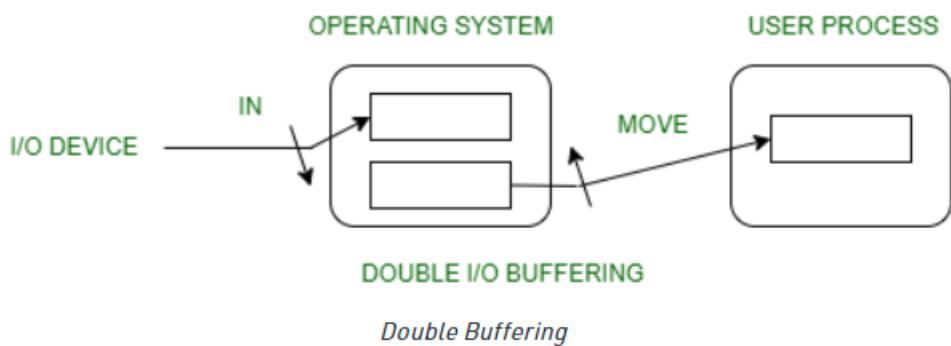


Figure 4.4: schéma double buffering

2. La "DMA" pour "Direct Memory Acces" est une interface mémoire qui permet de transférer à très haute performance des données entre les périphériques et la mémoire.

4.3.1 Traitement d'interruption DMA et double buffering

L'idée est de traiter les données du buffer DMA grâce à l'audio task. Des que le NVCI transmet une interruption du DMA (buffer Half ou Full) On exécute le forward de notre modèle de séparation de source sur la partie du buffer qui est pleine pendant que l'autre est entrain de chargé. Le résultat de la séparation est enregistré dans la SDRAM.

On a ainsi le schéma l'implémentation voir fig 4.5:

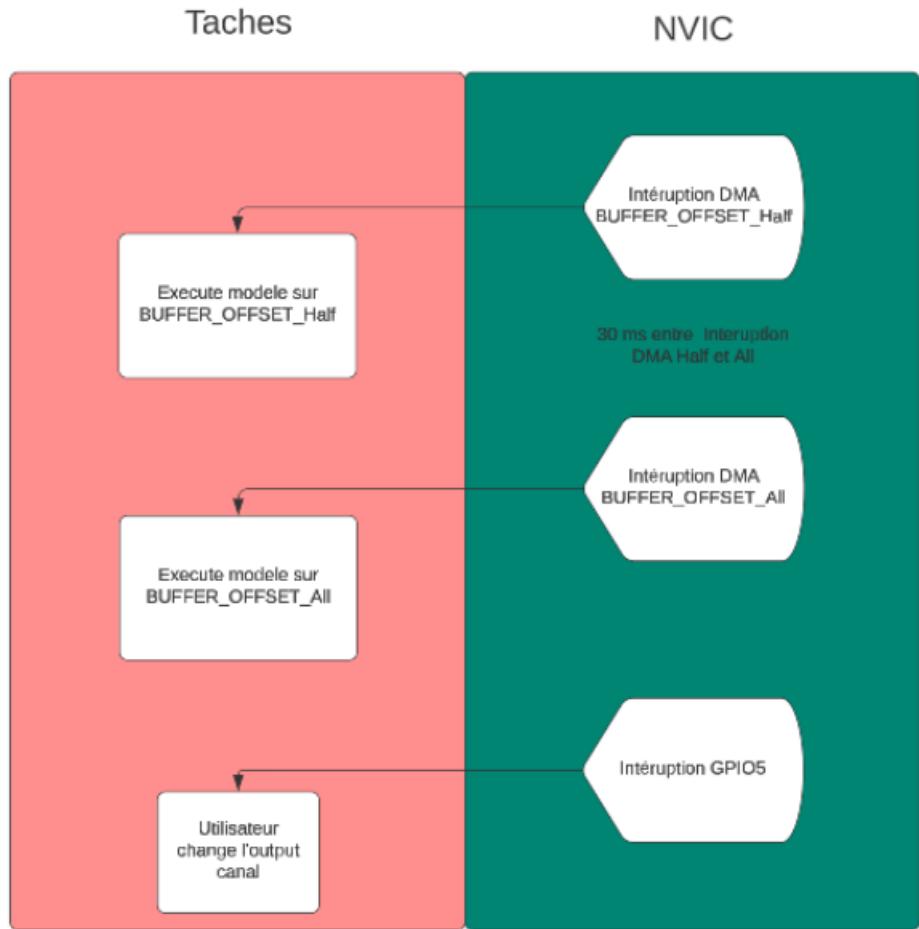


Figure 4.5: Interruption

Il est essentiel que l'audio processing soit exécuté en moins de ($dT = 30 \text{ ms}$) pour garantir l'exécution en temps réel. De plus on sait que le signal d'input est échantillonné à 11kHz ainsi la taille du buffer est calculé pour qu'un demi buffer soit rempli en 30 ms:

$$SizeDemiBuffer = dT * FreqEchantillonage = 0.030 * 11025 = 330 \quad (4.1)$$

$$SizeBufferDMA = 2 * SizeDemiBuffer = 660 \quad (4.2)$$

Le bufferDMA est ainsi de taille 660 échantillons float. On sait que notre modèle traite l'information sur 5ms, ainsi il faut que 6 itérations de notre modèle soient exécutées en moins de 30 ms pour maintenir une exécution temps réel. (55 échantillons doivent être traités en moins de 5 ms).

4.4 Traitement du signal, implémentation du modèle

Pour implémenté notre modèle, on utilise le package X cube AI. Cette librairie permet d'implémenter facilement un modèle.

4.4.1 Méthode d'implémentation de notre modèle

Voir également la dataSheet de IA package sur [8]

Nous présentons ici les étapes pour implémenter un modèle. Malheureusement par manque de temps, nous n'avons pas pu convertir notre modèle en keras ainsi j'utilise un modèle de teste qui prend en entrée une image de taille 90*3 (float) et effectue une classification sur 6 classes.

Ce modèle nous permet de tester l'implémentassions et l'utilisation du package IA.

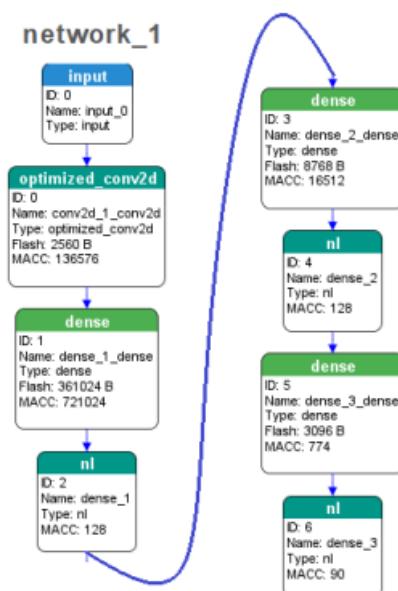
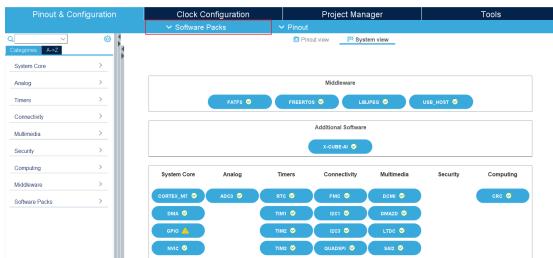


Figure 4.6: Réseau de Test

Voici ces étapes:

- Dans Stm32 cube MX sélectionné le gestionnaire de software Package -> select components On active le système core. Cela permet d'activer le software IA Dans cette partie, nous n'avons pas activé l'application. Ainsi nous devrons manuellement définir notre application (écrire le code d'utilisation de notre modèle). voir 4.7a et 4.7b
- La deuxième étape est de configurer notre modèle. Pour cela il suffit de se diriger dans la sous partie "Pinout and config" -> "Software Packs" -> "STMicroélectronics X-cube-AI". voir 4.9 indice 1

(a) Step1



(b) Step2

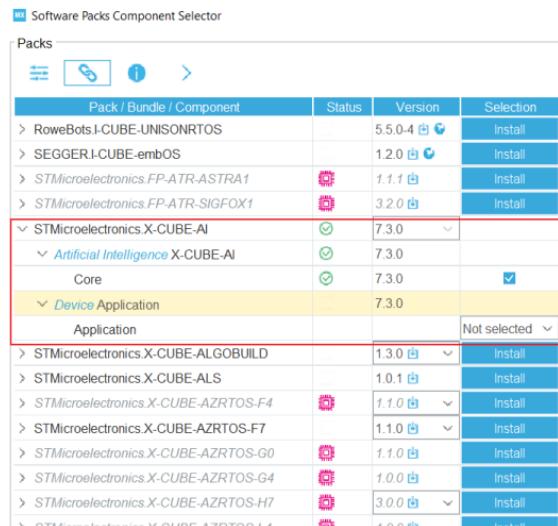


Figure 4.7: Activation de X-cube-AI

Delà, on peut télécharger notre modèle et l'analyser.

Voici un bref résumé de l'interface de contrôle de modèle

1. Lien vers l'interface de contrôle X - cube -IA.
2. Nom du modèle (ce nom sera utilisé pour caractériser l'ensemble des fichiers générés par cube MX pour le modèle).
3. Le type de Modèle.
4. La compression du modèle : ce paramètre permet de modifier le type des poids pour enregistrer le modèle sous la forme de int, float, double par exemple. Cela permet de réduire le stockage du modèle au détriment de ses performances.
5. L'optimisation peut être faite soit pour limiter l'impact sur la RAM soit pour augmenter les performances. Dans notre cas on utilise une implémentassions balancé car on a besoins à la fois de bonnes performances et d'un impact limité sur la mémoire.
6. Cette option permet de donner un fichier d'input "outputLabel" qui permet de tester notre modèle.
7. Show graph permet d'observer notre modèle sous la forme d'un graphique.
8. Analyse permet de déterminer automatiquement les besoins en mémoire de notre modèle. En effet lorsque l'on exécute le modèle sur la carte, cette dernière génère une instance modèle qui nécessite de l'espace mémoire dans la heap. Ainsi cette

près analyse nous donne un ordre d'idée de la mémoire à réserver pour notre modèle. voir 4.8

```
macc=875,232 weights=375,448 act=25,600 ram_io=0
Creating txt report file C:\Users\Antoine.Z\stm32cubemx\network_output\network_1_analyze_report.txt
elapsed time (analyze): 3.217s
Getting Flash and Ram size used by the library
Model file: model.h5
Total Flash: 396404 B (387.11 KiB)
Weights: 375448 B (366.65 KiB)
Library: 20956 B (20.46 KiB)
Total Ram: 28276 B (27.61 KiB)
Activations: 25600 B (25.00 KiB)
Library: 2676 B (2.61 KiB)
Input: 1080 B (1.05 KiB included in Activations)
Output: 24 B (included in Activations)

Done
Analyze complete on AI model
```

Figure 4.8: Résultat analyse

9. "Validate on desktop" permet d'exécuter le modèle sur l'ordinateur sans compression.
10. "Validate on target" permet d'exécuter le modèle sur la carte avec compression. On peut alors observer les différences de performance entre les deux implémentations (ordinateur et carte).

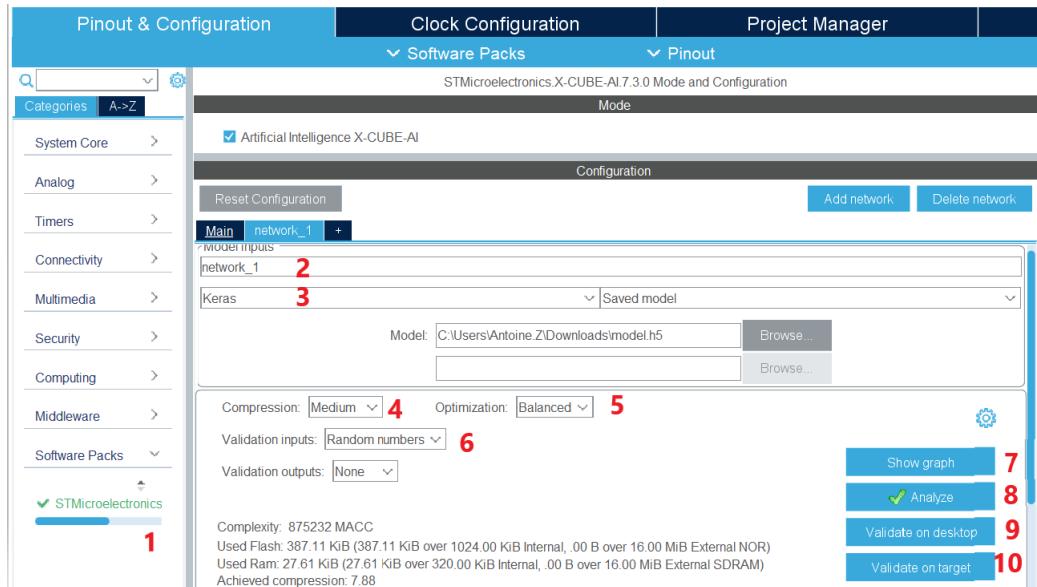


Figure 4.9: Configuration IA

- Dans cette étape on active la heap et la stack dans "System Core" -> "Cortex M7" (enable CPU Icache et CPU DCache)

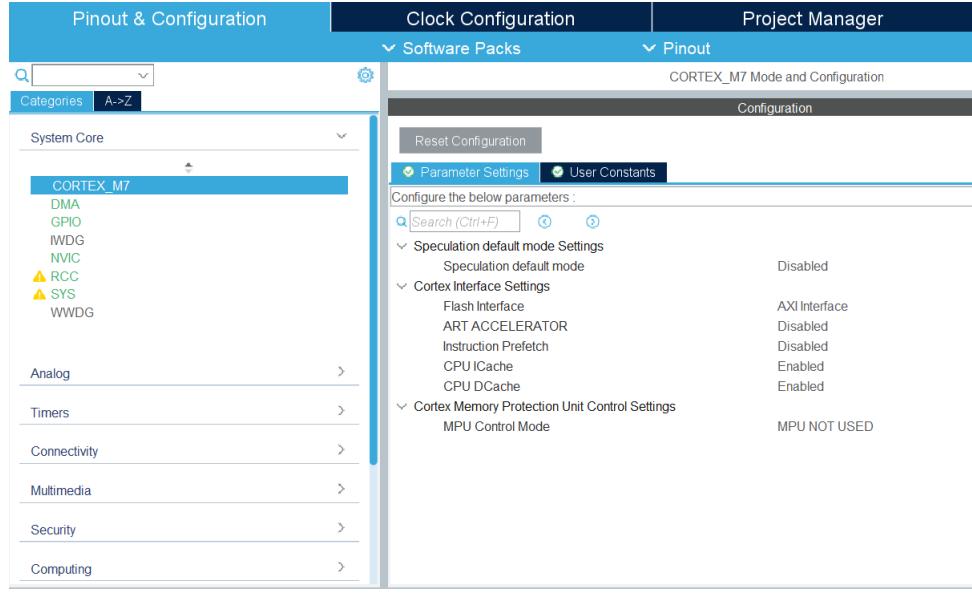


Figure 4.10: Configuration System Core

- On augmente la fréquence de fonctionnement du cortex au maximum pour augmenter la vitesse d'exécution du modèle :

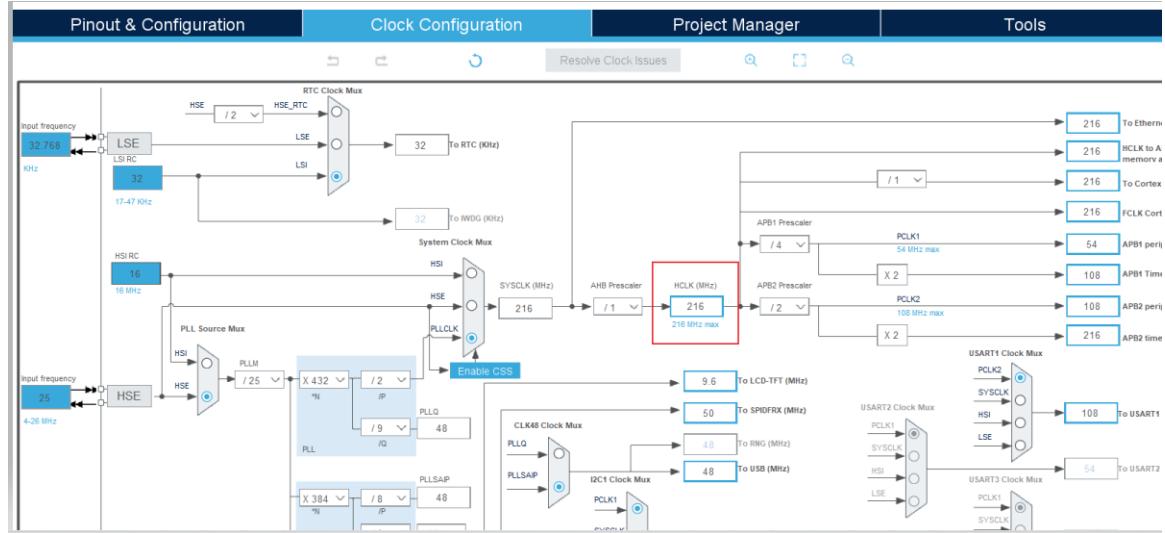


Figure 4.11: Gestion des Horloges

- Grâce aux résultats de l'analyse du modèle (Step 3) on peut ajuster la mémoire heap

pour que le modèle est suffisamment de mémoire pour être exécuté. Pour le modèle de test: voir fig 4.12:

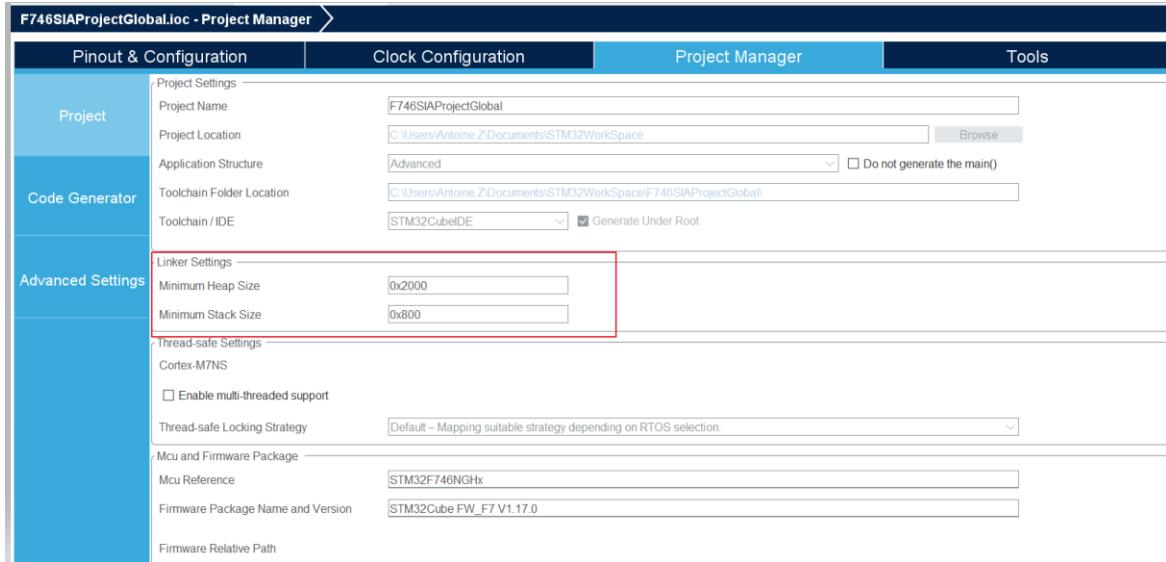


Figure 4.12: Augmentation de la heap mémory

- Finalement grâce à STM32 cube MX, on peut générer le code.

4.4.2 Le code, création de l'application du modèle

Dans cette étape on utilise le code générer par Stm32 cube MX et on le modifie pour définir notre application. Ainsi la première étape est de se familiariser avec les fichiers générés: Il y a deux fichiers qui nous intéressent:

1. `network_1_data_params.c`: Ce fichier comporte de les définitions des fonctions et des variables globales pour implémenter notre modèle (size Input, size output, run modèle , creat instance).
2. Le fichier `network_1_data_params.h` : Ce fichier comporte les poids du modèle.

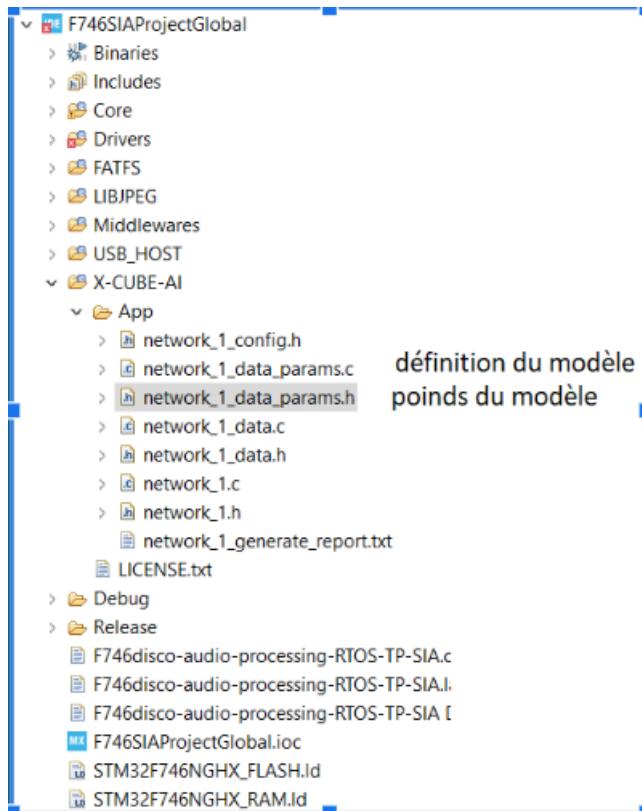


Figure 4.13: arborescence

Pour implémenté notre modèle il suffit de suivre les étapes suivante :

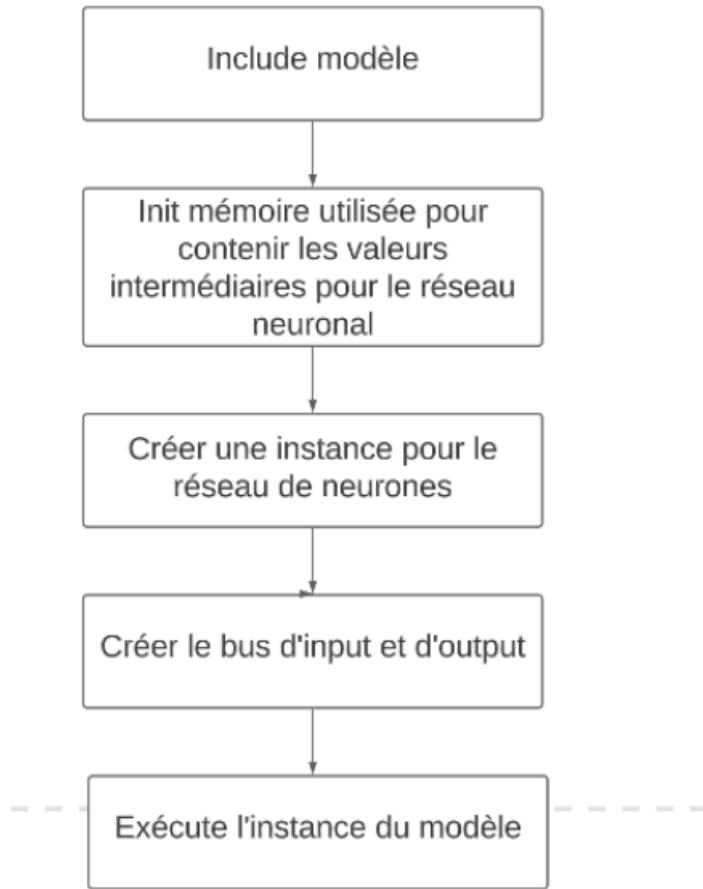


Figure 4.14: Étape de notre application

Pour plus d'information voir [7]

4.5 Résultat Hardware

Nous avons réussi à implémenter un modèle keras sur une carte STM32 (voir code projet : Project_modeleTest_sur_STM32) Lors de cette première implémentation, nous avons réglé les problèmes de compatibilités entre le package X-cube-IA, STM32-cube-Mx. Par la suite nous avons crée notre propre application pour le modèle de test. Et nous avons réussi à exécuter le modèle sur la carte.

Pour l'implémentation du projet de séparation de source voir projet F746_SIA_Project_Separation. Cette implémentation est loin d'être terminé. En effet il reste de nombreuses problématiques non résolus:

1. Une multitude d'erreurs de mémoire liées à la taille du modèle et à la gestion de mémoire par le RTOS.

2. le modèle n'est pas encore dans le bon format (Keras). Et n'est donc pas encore implémenté grâce au pacquage X-cube-IA.
3. l'optimisation du programme et la suppression des périphériques inutiles. Pour libérer de la place mémoire.

Pour la suite de ce projet, je pense qu'il serait intéressant de continuer l'implémentation de notre modèle sur STM32 mais également effectuer cette implémentations sur Zynq.

Bibliography

- [1] Xilinx,datasheet Zynq702, <https://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html>
- [2] ST,datasheet STM32F746NG, <https://www.st.com/en/evaluation-tools/32f746gdiscovery.html> sample-buy
- [3] Stefani, Domenico Peroni, Simone Turchet, Luca. (2022). A Comparison of Deep Learning Inference Engines for Embedded Real-time Audio Classification.
- [4] Editor(s): Jen-Tzung Chien, Source Separation and Machine Learning, Academic Press, 2019, ISBN 9780128177969, <https://doi.org/10.1016/B978-0-12-804566-4.00008-5.> (<https://www.sciencedirect.com/science/article/pii/B9780128045664000085>)
- [5] Luo, Yi and Mesgarani, Nima <https://arxiv.org/abs/1711.00541> TasNet: time-domain audio separation network for real-time, single-channel speech separation arXiv 2017 Creative Commons Attribution Non Commercial Share Alike 4.0 International
- [6] J. L. Roux, S. Wisdom, H. Erdogan and J. R. Hershey, "SDR – Half-baked or Well Done?," ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 2019, pp. 626-630, doi: 10.1109/ICASSP.2019.8683855.
- [7] Digi-Key ,TinyML: Getting Started with STM32 X-CUBE-AI | Digi-Key Electronics ,<https://www.youtube.com/watch?v=crJcDqlUbP4>
- [8] St, AI expansion pack for STM32CubeMX,<https://www.st.com/en/embedded-software/x-cube-ai.html>