Telecom Saint-Étienne
FISE 3

Co-Design
– Rapport de TP –

Étude en Filière Ingénieur sous Statut Étudiant
28.01.2026

Bastien DESCOS

Table des Matières

I. Notebook exemple cybersécurité

1. Présentation du dataset

Le dataset que nous allons utiliser est le UNSW-NB15, ce dataset a initialement 2,540,044 entrées réparties en 4 fichiers principaux. Nous allons utiliser 175,341 enregistrements en tant que base de données d'entraînement ainsi que 82,332 en tant que base de données de test.
Il contient différents types d'attaques permettant d'entraîner le modèle pour reconnaître ces attaques:

- Fuzzers
- Analysis
- Backdoors
- DoS
- Exploits
- Generic
- Reconnaissance
- Shellcode
- Worms

En l'occurrence, la version que nous téléchargeons est une version pré-quantifiée ce qui nous permettra un temps de téléchargement beaucoup plus court.

La sortie du réseau est simplement un retour qui indique si l'entrée est suspicieuse ou si elle est seine grâce à une probabilité allant de 0 (l'entrée est sûre) à 1 (l'entrée est suspecte).

2. Entrainement MLP

2.a) Modèle

Le réseau est un MLP avec 593 entrées, 3 couches cachées de 64 neurones, et une couche de sortie de 1 neurone. On retrouve donc 1 sortie qui indique si l'entrée est une attaque ou non.
Un modèle MLP (Multi-Layer Perceptron) est un réseau de neurones artificiels composé de plusieurs couches de neurones entièrement connectées. Chaque neurone dans une couche est connecté à tous les neurones de la couche suivante.
Cela nous fait un total de $593\times64 + 64\times64 + 64\times64 + 64\times1 = 38{,}081$ poids.
Le modèle est entraîné avec une fonction de perte binaire cross-entropy et l'optimiseur Adam. L'entraînement est effectué sur 10 époques avec un batch size de 256.

2.b) Quantification

Nous utilisons une quantification pour nos poids ainsi que nos ReLu de 2 bits.

Pour la quantification, il exite plusieurs méthodes: QAT (quantization-aware training) et PTQ (post-training quantization). La première méthode consiste à quantifier durant l'entraînement, la seconde consiste à quantifier après l'entraînement. La quantification utilisée dans notre projet est une quantification QAT.

2.c) FINN

Le framework FINN est un framework open-source développé par Xilinx pour la conception et le déploiement de réseaux de neurones quantifiés sur des FPGA. Il sert d'interface entre le ONNX et Vitis. Il nous permet donc de ne pas avoir à créer le HLS à la main et donc nous simplifie grandement la tâche.
FINN utilise dans notre cas une quantifiaction binaire pour les poids et les activations, ce qui permet de réduire considérablement la taille du modèle et d'améliorer la vitesse d'inférence sur le FPGA.
Le modèle quantifié est ensuite converti en une représentation compatible avec le matériel FPGA à l'aide de FINN, qui génère du code HDL (Hardware Description Language) pour l'implémentation sur le FPGA.
FINN offre également des outils pour l'optimisation du modèle, la génération de bitstreams pour le FPGA.

Pour l'importation dans FINN, on ajoute 7 zéros afin de passer d'un nb premier 593, à un nb facilement découpable (600): W˙new = np.pad (W˙orig, [(0,0), (0,7)]).

Pour que FINN fonctionne, il nécessite une quantification binaire codée entre {-1, +1}, c'est pourquoi nous avons dû adapter notre modèle Brevitas pour qu'il corresponde à cette contrainte. Pour ce faire nous avons utilisé un wrapper Brevitas qui convertit les poids et les activations de {0, 1} à {-1, +1}.

FINN sort beaucoup de fichiers de sortie, certains peuvent être très imposant comme le bitfile, c'est pourquoi il y a des paramètres afin de gérer les fichiers de sortie:

- **ESTIMATE_REPORTS**: fourni les rapports des ressources attendues et des performances par couche et pour l'ensemble du réseau sans synthèse Vivado complète;
- **STITCHED_IP**: crée un design IP stream-in stream-out qui peut être intégré dans d'autres designs Vivado IPI ou RTL;
- **RTLSIM_PERFORMANCE**: utiliser PyVerilator pour effectuer un test de performance/latence du design STITCHED˙IP;
- **OOC_SYNTH**: exécuter une synthèse hors contexte (juste l'accélérateur lui-même, sans aucun système l'entourant) sur le design STITCHED˙IP pour obtenir les ressources FPGA post-synthèse et la fréquence d'horloge réalisable;
- **BITFILE**: intégrer l'accélérateur dans un shell pour produire un bitfile autonome;
- **PYNQ_DRIVER**: générer un pilote Python PYNQ qui peut être utilisé pour lancer l'accélérateur;
- **DEPLOYMENT_PACKAGE**: créer un dossier contenant les sorties BITFILE et PYNQ_DRIVER, prêt à être copié vers la plateforme FPGA cible.
- **OUTPUT_DIR**: indique le dossier dans lequel l'ensemble des sorties du programmes seront écrites.
- **STEPS**: indique la liste des étapes prédefinie ou personnalisée que FINN va faire pour le build de l'accélérateur.

Pour le déploiement sur la carte nous aurons besoin du BITFILE ainsi que du PYNQ_DRIVER. L'ESTIMATE_REPORTS peut être utile en amont pour savoir les ressources et les performances de notre accélérateur.

3. Comparaison des performances

Nous allons faire varier la quantification des poids et des activations de 2 bits à 32 bits et comparer les performances du modèle sur le dataset UNSW-NB15.
Nous allons mesurer la précision (accuracy) ainsi que le temps d'inférence sur CPU.
Voici les résultats obtenus:

Modèle	MLP	MLP	MLP	MLP	MLP
Nb quantification	2	4	8	16	32
Accuracy (%)	73.5	79.2	81.3	82.1	82.3
Temps CPU (s)	12.34	13.45	14.56	15.67	16.78
Temps Inférence (s)	0.001234	0.001345	0.001456	0.001567	0.001678
Temps inférence (ms)	1.234	1.345	1.456	1.567	1.678
Nb Inférences/s	810	743	686	638	596

Comme nous pouvons le voir, la quantification a un impact significatif sur les performances du modèle. En effet, plus la quantification est faible, plus la précision diminue. Cependant, le temps d'inférence diminue également, ce qui peut être bénéfique pour les applications en temps réel. Nous remarquons aussi que la diminution de la précision n'est pas linéaire par rapport à la taille de la quantification. En effet, dès que la quantification est de 4 bits, nous observons une accuracy qui est très semblable.
Il est donc important de trouver un compromis entre la taille de la quantification et les performances du modèle en fonction des besoins de l'application.

4. Estimations implémentations FINN

Les max FPS visés sont de 1 000 000 (1 MOps). La période est de 10ns soit une fréquence de 100 000 000 Hz, soit 100MHz.
En baissant TARGET˙FPS, nous pouvons réduire la configuration matérielle nécessaire.
Coté ressource, le programme estime les utilisations suivantes:

- LUT: 9354
- BRAM_18K: 45

Les performances estimées sont de 1 562 500,0 FPS.

L'architecture comporte les élements suivants:

- "PE": 16 + 1 + 1 +1 = 19
- "SIMD": 40 + 64 + 64 + 1 = 169

B. Partie Stitched IP et PYNQ bitfile and Driver

Estimation par FINN:

- LUT: 17640 + 1096 + 1042 + 268 = 20046
- BRAM_18K = 0
- FF: 2037 + 801 + 803 + 68 = 3909

- DSP = 32 + 32 + 1 = 65

Les performances estimées par FINN sont de 448430.49327354255 FPS.

Les FIFOS permettent de passer les informations d'une couche à une autre ainsi que de synchroniser l'ensemble. Ici elles sont de 32 à 2 entre la couche d'entrée et le couche 1 puis de 2 à 2 entre les autres couches. Cela veut donc dire que les couches ont besoin d'un certain nombre d'entrées pour pouvoir fonctionner (ici 2).

5. Synthèse Vivado

Nous pouvons maintenant regarder les ressources utilisées après synthèse Vivado. Pour les lire, nous nous sommes rendus dans le rapport post synthèse Vivado qui est parmi les rapports dans le final output du programme 3. Nous obtenons les résultats suivants:

- LUT: 12711
- BRAM_18K: 2
- FF: 15854
- BRAM_36K: 22
- DSP: 129
- SRL: 381

Pour ouvrir Vivado, nous devons definir l'environnement avec la commande suivante en étant à la racine:

```
source /opt/Xilinx/Vivado/2020.2/settings64.sh
```

Puis nous ouvrons Vivado en étant dans le dossier /tmp et dans le sous dossier créé par le programme 3 avec la commande:

```
vivado finn_zynq_link.xpr
```

Les valeurs indiqués dans le rapport post-synthèse sont également retrouvable au sein de Vivado dans les ressources implémentées:

Resource	Utilization	Available	Utilization %
LUT	12711	53200	23.89
LUTRAM	1013	17400	5.82
FF	15854	106400	14.90
BRAM	23	140	16.43
DSP	129	220	58.64
BUFG	1	32	3.13

Figure 1. Tableau de l'utilisation des ressources

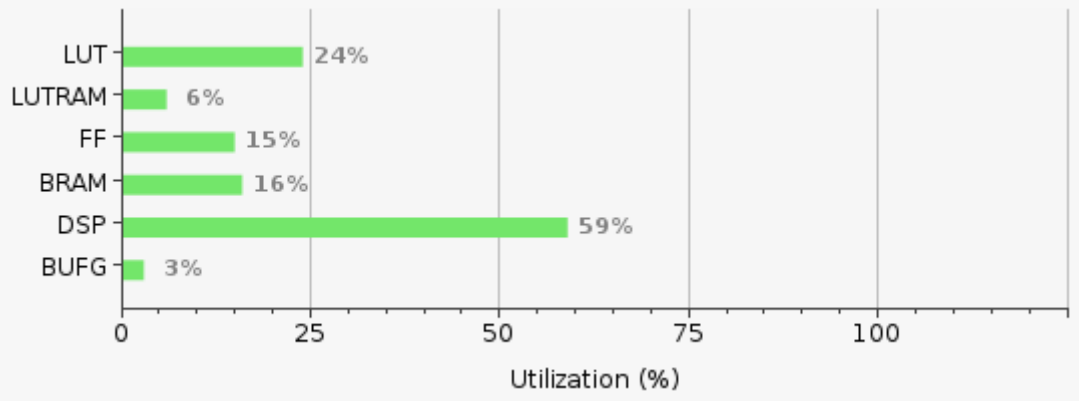


Figure 2. Graphique de l'utilisation des ressources

Comme nous pouvons le voir, l'utilisation des ressources est très faible par rapport aux ressources disponibles sur la carte FPGA. On voit que les ressources utilisées sont largement inférieures aux ressources estimées par FINN. Cela peut être dû à plusieurs facteurs, notamment les optimisations effectuées par le synthétiseur Vivado qui peuvent réduire l'utilisation des ressources par rapport aux estimations initiales de FINN.

La répartition des ressources utilisées est la suivante:

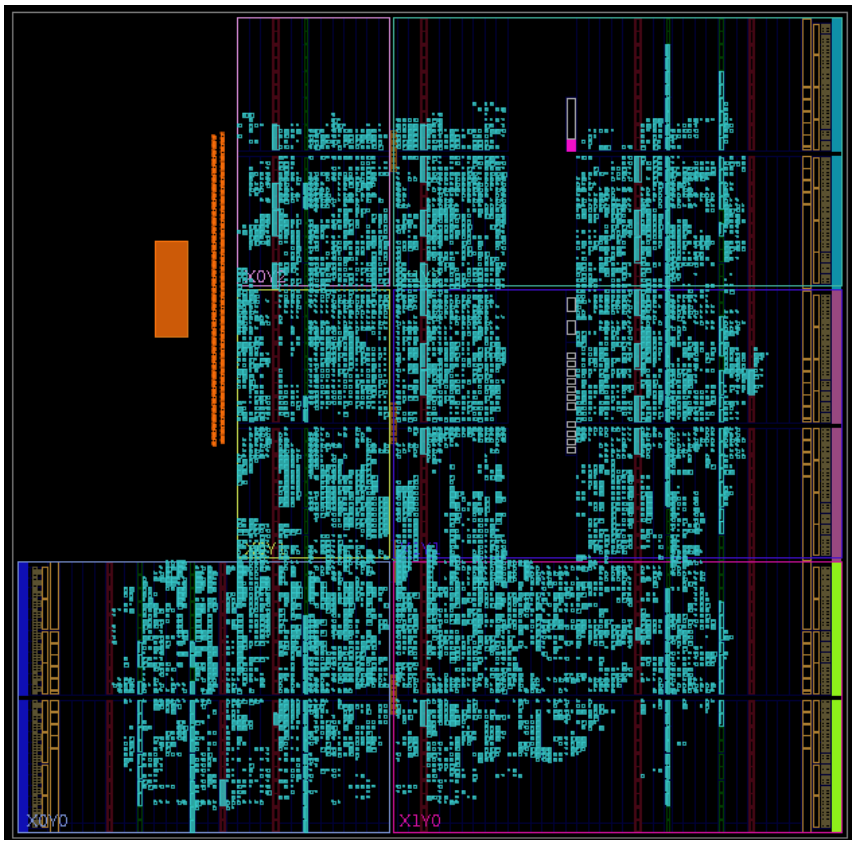


Figure 3. Image de la répartition des ressources utilisées

Nous pouvons croire que la majorité des ressources sont utilisées mais en réalité, nous voyons bien dans la table ainsi que dans le graphique que l'utilisation des ressources est très faible par rapport aux ressources disponibles sur la carte FPGA. La présence de bleu de partout dans l'image explique juste un grand étalement des ressources utilisées sur la carte.

6. Carte

Partie non réalisée en TP, faite à partir du screenshot fourni.

Après avoir généré le bitfile et le driver PYNQ, nous pouvons maintenant déployer notre modèle sur la carte PYNQ-Z2.
Pour ce faire, nous copions le dossier de déploiement généré par FINN sur la carte PYNQ-Z2. Ensuite, nous utilisons le driver PYNQ pour charger le bitfile sur la carte et exécuter des inférences sur le modèle.

```
Final accuracy: 91.840000
root@pynq:/home/xilinx/jupyter_notebooks/mlp-mnist/driver# python3.10 driver.py --exec_mode throughput_test ../bitfile/
finn-accel.bit          finn-accel-errorBN.bit  finn-accel-errorBN.hwh  finn-accel.hwh          .ipynb_checkpoints/
root@pynq:/home/xilinx/jupyter_notebooks/mlp-mnist/driver# python3.10 driver.py --exec_mode throughput_test ../bitfile/finn-ac
cel.bit --batchsize 5000
usage: driver.py [-h] [--exec_mode EXEC_MODE] [--platform PLATFORM] [--batchsize BATCHSIZE] [--device DEVICE]
                 [--bitfile BITFILE] [--inputfile [INPUTFILE ...]] [--outputfile [OUTPUTFILE ...]]
                 [--runtime_weight_dir RUNTIME_WEIGHT_DIR]
driver.py: error: unrecognized arguments: ../bitfile/finn-accel.bit
root@pynq:/home/xilinx/jupyter_notebooks/mlp-mnist/driver# python3.10 driver.py --exec_mode throughput_test --bitfile ../bitfi
le/finn-accel.bit --batchsize 5000
Results written to nw_metrics.txt
root@pynq:/home/xilinx/jupyter_notebooks/mlp-mnist/driver# cat nw_metrics.txt
{'runtime[ms]': 5.808830261230469, 'throughput[images/s]': 860758.4961418486, 'DRAM_in_bandwidth[MB/s]': 674.8346609752093, 'D
RAM_out_bandwidth[MB/s]': 17.21516992283697, 'fclk[mhz]': 100.0, 'batch_size': 5000, 'fold_input[ms]': 0.148773193359375, 'pac
k_input[ms]': 0.09775161743164062, 'copy_input_data_to_device[ms]': 25.090694427490234, 'copy_output_data_from_device[ms]': 0.
7543563842773438, 'unpack_output[ms]': 32159.178972244263, 'unfold_output[ms]': 0.09465217590332031}root@pynq:/home/xilinx/jup
yter_notebooks/mlp-mnist/driver#
```

Figure 4. Image de l'exécution du modèle sur la carte PYNQ-Z2

Nous remarquons plusieurs informations intéressantes:

- L'accuracy indiquée est de 91.84%. Nous pouvons cependant noter que le dossier s'appelle mlp-mnist, ce qui indique que le dataset utilisé est MNIST et non UNSW-NB15. Cela peut expliquer l'accuracy élevée.
- Le temps d'éxécution est de 5.8088 ms, sachant que le batch size est de 5000, cela nous donne un temps d'inférence par image de 1.16176 µs, soit 860 758 inférences par seconde.
- La fréquence d'horloge est de 100 MHz, ce qui est conforme à nos attentes.

Grâce au screenshot vivado, nous remarquons que même si le dataset est différent,le modèle lui comporte une configuration similaire à la notre (PE et SIMD identiques). Nous pouvons donc en conclure que notre modèle devrait avoir des performances similaires.

II. Modification MLP pour CIFAR10

1. CIFAR10 dans FINN

Le dataset CIFAR10 est un ensemble de données utilisé pour la reconnaissance d'images. Il contient 60,000 images d'entraînement et 10,000 images de test, chaque image étant une image en couleur de 32×32 pixels représentant un objet appartenant à l'une des 10 classes suivantes: avion, automobile, oiseau, chat, cerf, chien, grenouille, cheval, bateau, camion.

Afin de nous servir de CIFAR10, nous allons utiliser à nouveau le framework FINN en modifiant les paramètres de configuration pour qu'ils soient adaptés au nouveau dataset. Nous allons également devoir modifier le modèle de réseau de neurones pour qu'il soit adapté à la reconnaissance d'images.

2. Impact de la quantification

Afin de réaliser la quantification, nous allons utiliser Brevitas qui est une bibliothèque de quantification pour PyTorch. Pour ce faire, nous avons utilisé des couches de Brevitas pour remplacer les couches standards de PyTorch. Cela s'effectue comme suit:

```
import brevitas.nn as qnn
qnn.QuantConv2d(3, 6, kernel_size=5, bias=True, padding = 2, weight_bit_width=16)
```

Nous voyons dans cette ligne la définition d'une couche de convolution quantifiée avec des poids sur 16 bits (weight_bit_width=16). Le reste ne change pas par rapport à une couche de convolution standard de PyTorch.

Pour réaliser l'impact de la quantification, nous avons entraîné plusieurs modèles avec des poids et des activations de différentes tailles (2, 4, 8, 16, 32 bits) et nous avons comparé les performances de ces modèles sur le dataset CIFAR10. Nous avons également comparé ces modèles avec un modèle non quantifié (poids et activations en 32 bits flottants).
Pour les modèles utilisés nous avons utilisé un modèle simple de CNN avec 2 couches de convolution suivies de 2 couches fully connected type LeNet-5.
Nous avons également utilisé un modèle de MLP avec 3 couches soit 1 couche d'entrée (100 neurones), une couche cachée (50 neurones), et une couche de sortie (10 neurones).
Tout ces modèles ont été entraînés sur CPU et sur GPU pour comparer les performances, et voir si la quantifiaction avait un impact différent en fonction du moteur de calcul. Les métriques pour la mesure seront la précision (accuracy) ainsi que le temps d'inférence.

Modèle	CNN	CNN	CNN	CNN	CNN	CNN	CNN	MLP
Nb quantification	1	2	4	8	16	32	N/B	N/B
Accuracy (%)	10	56.2	61.2	63.6	64.9	64.4	63.5	41.7
Avg Loss	NaN	1.257583	1.112131	1.048991	1.024594	1.026724	1.046408	1.642302
Temps CPU (s)	4.574	3.6179	4.1767	5.34	4.0933	6.4669	4.5768	4.4393
Temps Inférence (s)	0.0004574	0.00036179	0.00041767	0.000534	0.00040933	0.00064669	0.00045768	0.00044393
Temps inférence (ms)	0.4574	0.36179	0.41767	0.534	0.40933	0.64669	0.45768	0.44393
Nb Inférences/s	2186	2765	2394	1872	2443	1545	2185	2253

Table 1. Tableau des performances CPU CIFAR10

Modèle	CNN	CNN	CNN	CNN	CNN	CNN	CNN	MLP
Nb quantification	1	2	4	8	16	32	N/B	N/B
Accuracy (%)	10	57	62.1	62.2	63.9	62.3	63.7	40.2
Avg Loss	NaN	1.232358	1.090406	1.084034	1.033	1.078849	1.058	1.67
Temps GPU (s)	3.698327	4.547411	3.6645641	3.685435	3.700761	4.67698526	1.601	1.73683
Temps Inférence (s)	0.0003698327	0.0004547411	0.00036645641	0.0003685435	0.0003700761	0.000467698526	0.0001601	0.000173683
Temps inférence (ms)	0.3698327	0.4547411	0.36645641	0.3685435	0.3700761	0.467698526	0.1601	0.173683
Nb Inférences/s	2704	2200	2729	2713	2702	2138	6246	5758

Table 2. Tableau des performances GPU CIFAR10

Nous voyons que la quantifiaction a un impact significatif sur les performances du modèle. En effet, plus la quantification est faible, plus la précision diminue. Une fois la quantification fait, il n'y a pas de grande différence entre les différentes tailles de quantification en terme de vitesse d'éxécution.
Nous observons cependant que la quantification a un impact plus important sur le GPU que sur le CPU. En effet, le temps de calcul sur GPU du modèle quantifié est grandement plus élevé que celui du modèle non quantifié, divisant jusqu'à par 3 les performances. Ceci s'explique par le fait que les GPU sont optimisés pour les calculs en virgule flottante, et que la quantification réduit la précision des calculs, ce qui peut entraîner une perte de performance sur ces architectures.
En revanche, sur CPU, la différence de performance entre le modèle quantifié et le modèle non quantifié est moins prononcée, car les CPU sont plus polyvalents et peuvent gérer efficacement

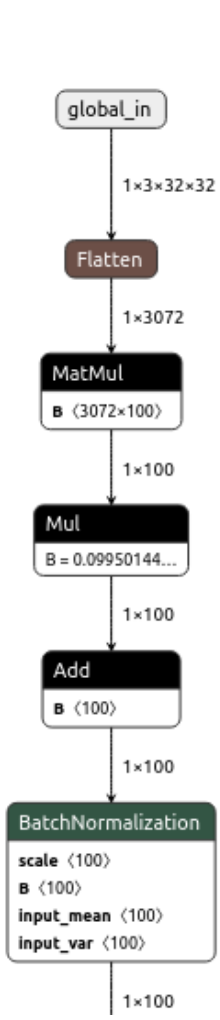
les calculs en virgule flottante et les calculs quantifiés.
Nous remarquons que lorsque la quantification est de 2 bits, la précision chute drastiquement, ce qui indique que cette taille de quantification est trop faible pour ce type de modèle et de dataset. Pire encore, la quantification binaire (1 bit) ne permet pas du tout d'apprendre, le modèle retourne une accuracy de 10% ce qui est équivalent à un tirage aléatoire.

Sur FPGA la quantification est très bénéfique car elle permet de réduire la taille du modèle et d'améliorer la vitesse d'inférence, ce qui est crucial pour les applications en temps réel.
L'amélioration du temps d'inférence s'explique par le fait que les opérations sur des entiers de faible taille sont beaucoup plus rapides et consomment moins de ressources que les opérations en virgule flottante, mais aussi par le fait que nous pouvons parralléliser les opérations sur FPGA.

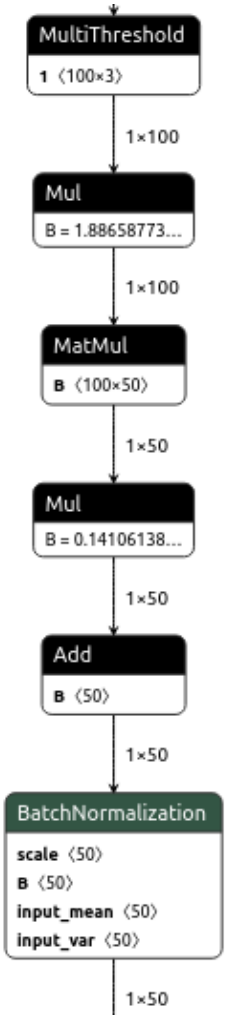
3. Performances estimées

3.a) Modèle Importé

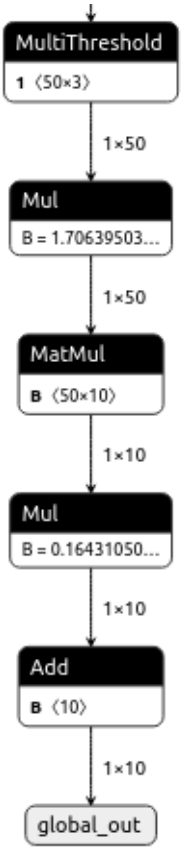
Voici le modèle importé:



(a) Légende image 1



(b) Légende image 2



(c) Légende image 3

Figure 5. Image du modèle importé dans Netron

L'accuracy lors du process sur le CPU est de 41,03% pour le MLP.

3.b) Importation dans FINN

Les max FPS visés ne sont pas les mêmes que dans la partie 1 (1 MOps), car après essai il semblait impossible qu'il puisse y arriver et créait donc une erreur lors de la création du bitfile. Cela semble logique, quand nous voyons la complexité du dataset CIFAR10 comparé à UNSW-NB15. De même, nous avons vu sur le sceenshot que la vitesse de sortie n'est que de 840 000 FPS pour un MNIST (peut-être pour un UNSW-NB15 car nous n'avons pas l'information du dataset utilisé pour le screenshot). Je suis donc parti sur 1MOps dans un premier temps pour le repport only et j'ai adapté pour mettre 100 000 FPS (100 kOps) pour la génération du bitfile et du driver.

Les performances estimées sont de 1 041 667 FPS soit quand même 521 000 FPS de moins que pour le dataset UNSW. Cela semble donc confirmer ce que nous disons dans le paragraphe précédent.
Nous voyons par contre que la vitesse est bien plus grande que pour les CPU et GPU, ce qui est normal dû à la parallèlisation des opérations sur FPGA.

Coté ressource, le programme estime les utilisations suivantes:

- LUT: 179483
- BRAM_18K: 186

L'architecture comporte les élements suivants:

- "PE": 100 + 5 + 1 = 106
- "SIMD": 32 + 25 + 10 = 70

B. Partie Stitched IP et PYNQ bitfile and Driver

Estimation par FINN:

- LUT: 10138 + 660 + 631 = 11429
- BRAM_18K: 0
- FF: 29097 + 545 + 209 = 30051
- DSP : 170 + 7 + 1 = 178

Les FIFOS ont une profondeur de 32 pour celle entre la couche d'entrée et la première couche de neurones puis de 2 entre les couches.

3.c) Synthèse Vivado

Nous pouvons maintenant regarder les ressources utilisées après synthèse Vivado. Pour les lire, nous nous sommes rendus dans le rapport post synthèse Vivado qui est parmi les rapports dans le final output du programme 3. Nous obtenons les résultats suivants:

- LUT: 19720
- BRAM_18K: 3

- FF: 40333
- BRAM_36K: 19
- DSP: 220
- SRL: 1188

Les valeurs indiqués dans le rapport post-synthèse sont également retrouvable au sein de Vivado dans les ressources implémentées:

Resource	Utilization	Available	Utilization %
LUT	19713	53200	37.05
LUTRAM	1818	17400	10.45
FF	40264	106400	37.84
BRAM	20.50	140	14.64
DSP	220	220	100.00
BUFG	1	32	3.13

Figure 6. Tableau de l'utilisation des ressources

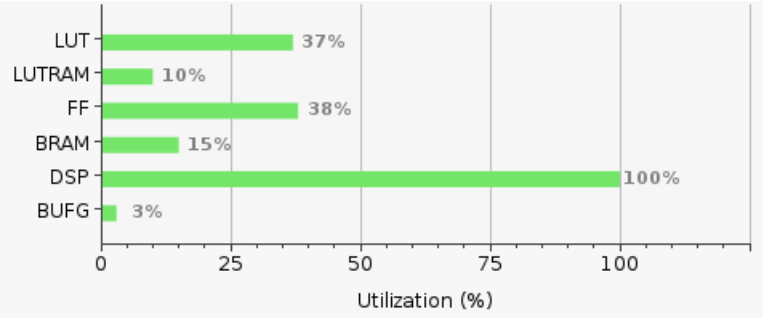


Figure 7. Graphique de l'utilisation des ressources

Comme nous pouvons le voir, l'utilisation des ressources est plus importante par rapport aux ressources utilisés par le MLP du dataset UNSW, ce qui est normal du à la taille plus importante du modèle. On voit que les ressources utilisées sont toujours largement inférieures aux ressources estimées par FINN. Nous voyons égalemment que les ressources utilisées commencent à être significatives par rapport aux ressources disponibles sur le SoC FPGA notamment en terme de DSP où nous utilisons les 220 DSP disponible sur la carte.

La répartition des ressources utilisées est la suivante:

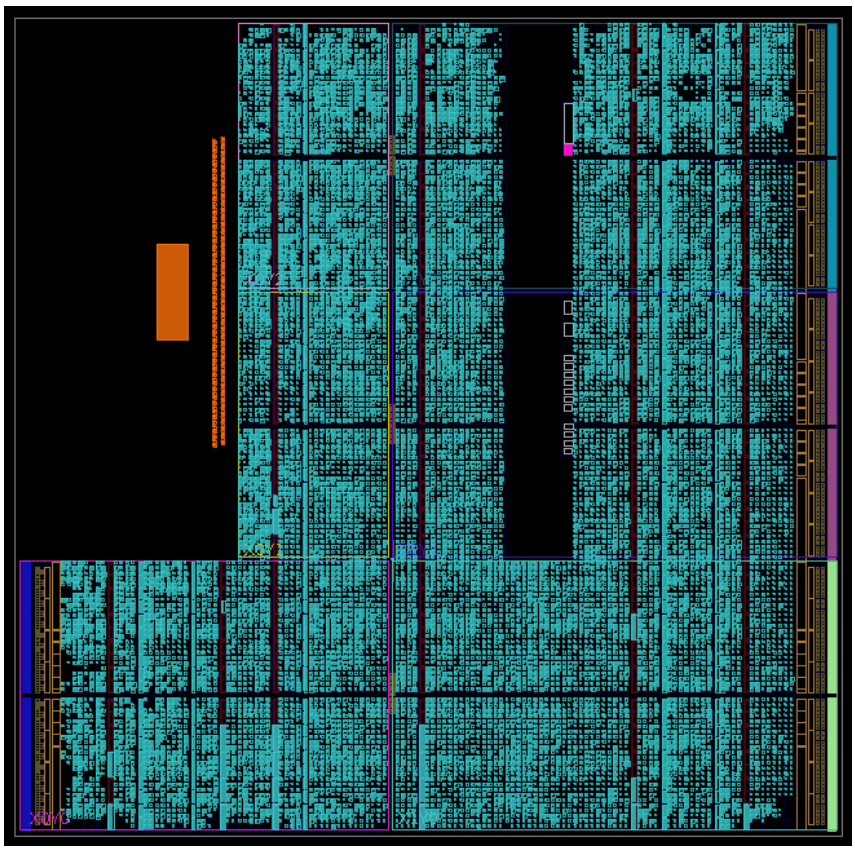


Figure 8. Image de la répartition des ressources utilisées

Nous voyons en effet que l'utilisation des ressources commence à atteindre certaines limites, ce qui est visible par la grande ligne bleue représentant les DSP. Même si nous sommes loin d'une saturation des ressources autre que DSP, nous voyons que la répartition est homogène à l'ensemble de la carte, ce qui a pour effet de paraitre saturé à tout les plans.

Il est à noter que lors de l'implementation du modèle sur la carte, nous avons rencontré un critical warning nous indiquant que le design ne parvient pas à atteindre les timings requirements.

Figure 9. Message du Critical Warning

III. Bibliographie

Liens vers les références utilisées pour la réalisation du rapport:
`https://www.kaggle.com/code/mrwellsdavid/unsw-nb15-dataset-mlp-classifier/notebook`
`https://xilinx.github.io/brevitas/v0.12.1/tutorials/tvmcon2021.html`
`https://github.com/Xilinx/brevitas?tab=readme-ov-file`
`https://arxiv.org/pdf-/2103.13630`
`https://arxiv.org/pdf/1612.07119`
`https://www.youtube.com/watch?v=zw2aG4PhzmA`