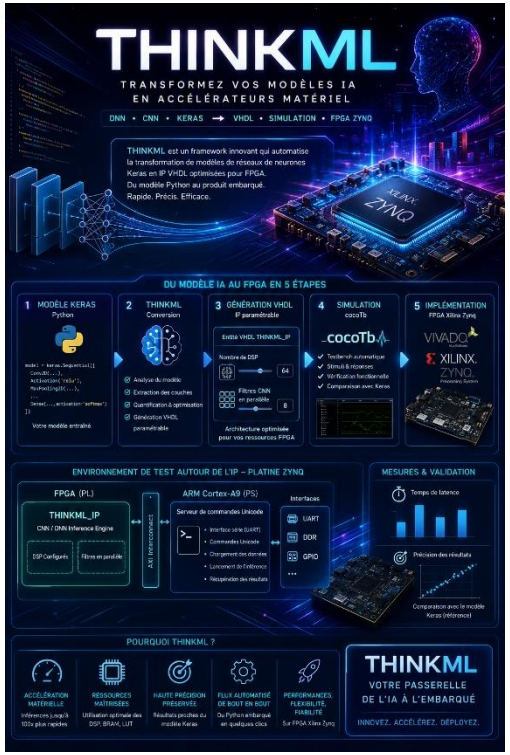
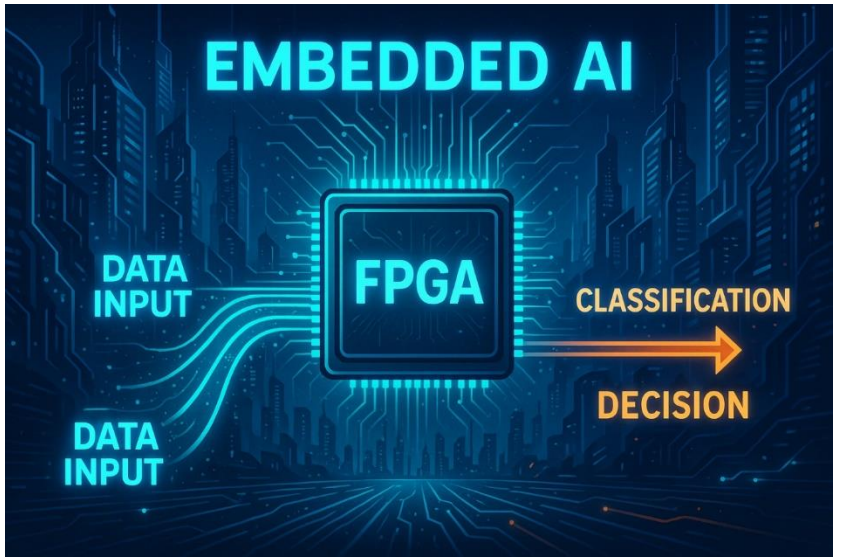
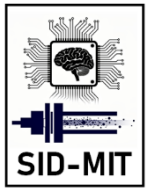
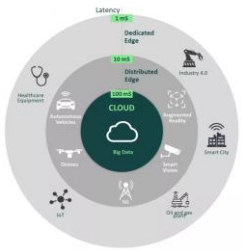
Frédéric Druillole & Guillaume Maraninchi (PFE Enseirb-Mathmeca)

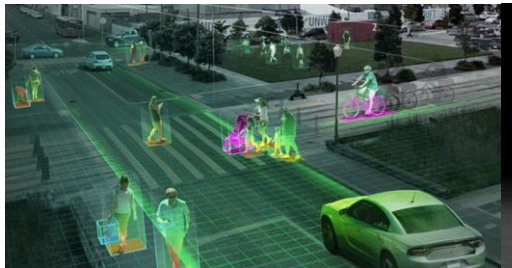
THINKML: Optimisation de l'inférence de réseaux de neurones pour FPGA AMD-Xilinx

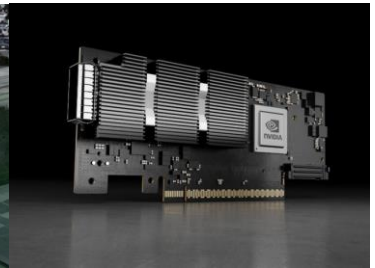


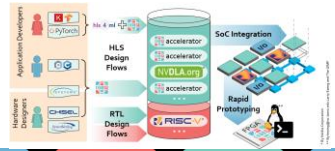


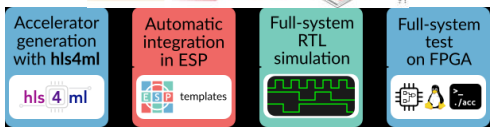










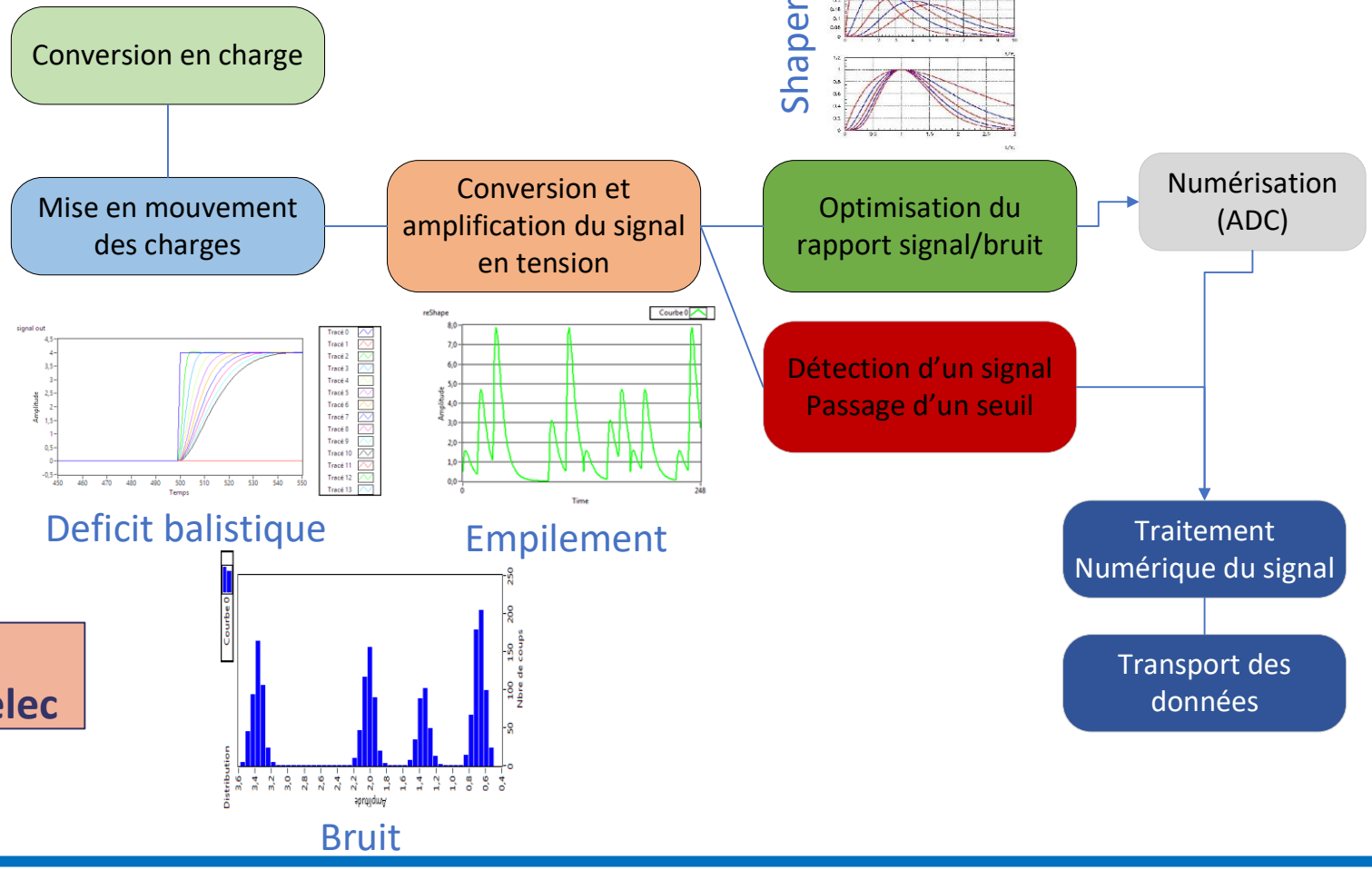


Le traitement du signal classique des détecteurs capacitifs

Architectures fonctionnelles

Les erreurs de mesures irrésolues

- **L'empilement**
- **Le déficit balistique**
- **Le cross-talk**
- **Le bruit du détecteur ou du capteur**
- **Le bruit de la chaîne électronique de mesure**

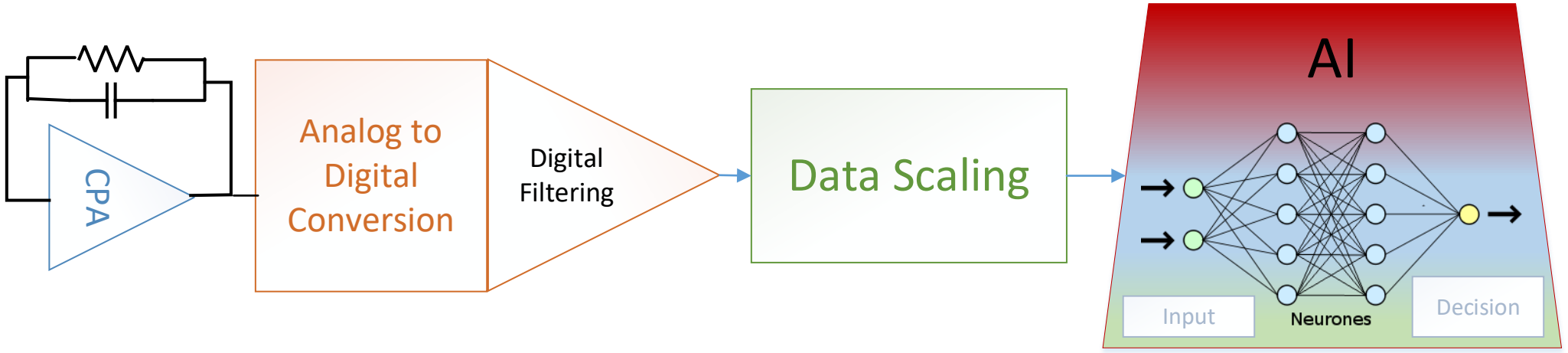


$$\sigma^2_{total} = \sigma^2_{det} + \sigma^2_{pu} + \sigma^2_{ba} + \sigma^2_{elec}$$

L'électronique est alors optimisée pour une seule application, un type de détecteur (Semi-conducteur, Det. Gazeux, PMT,..)

Architecture de base

Chaîne de mesure numérique minimisant la partie analogique



➔ TNS pour avoir une mesure parfaite ➔ Utilisation de modèles de Reseaux de Neurones (MLP, CNN, RNN)

Performances attendues

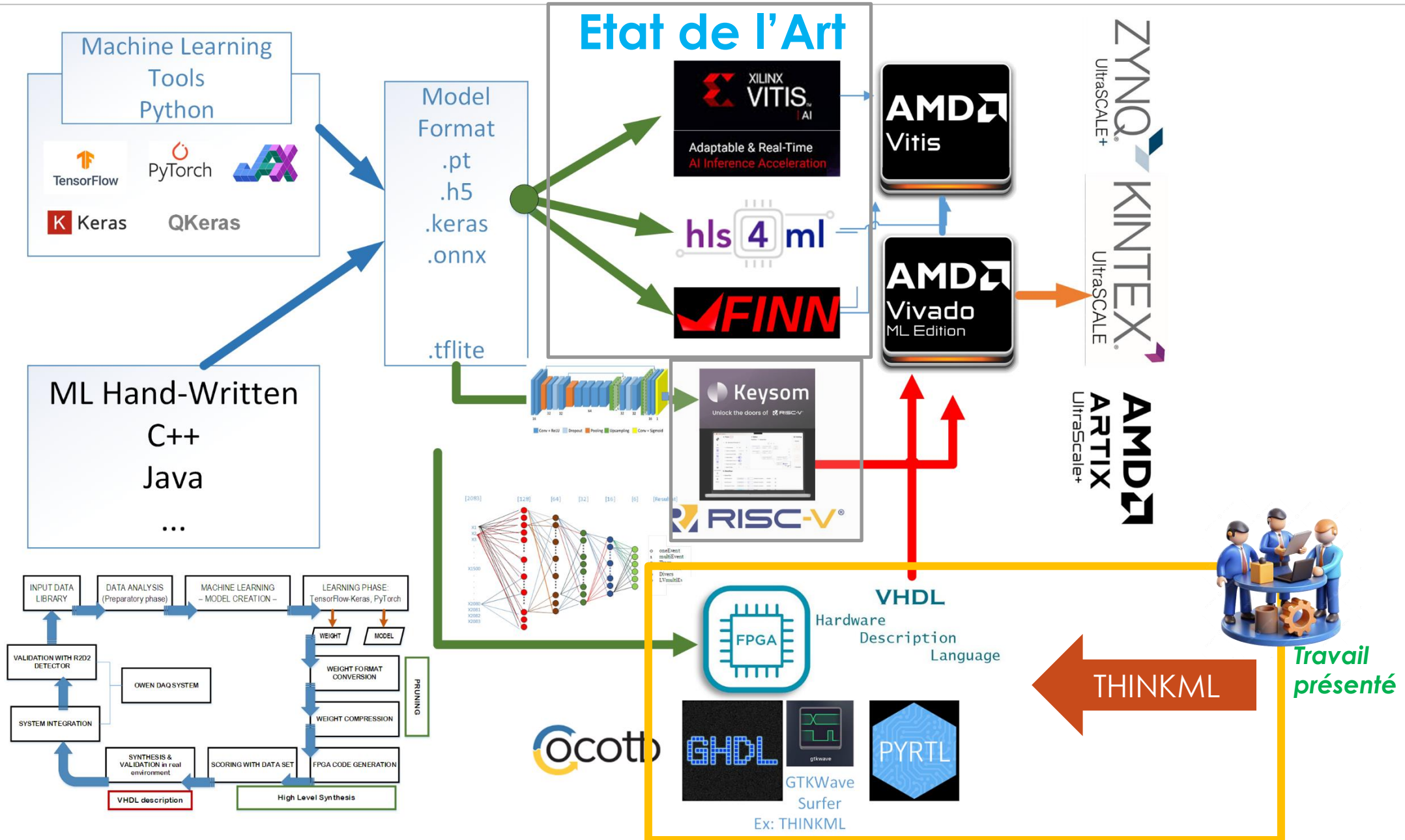
- **Gestion des Ressources FPGA versus Temps de calcul:**
  - **Nbre de DSP**
  - **Parallelisme des operations**
- **Gestion de la Latence versus Ressources FPGA**

➔ **Modèles sur petit FPGA Artix-7, Zynq-xcz7020 ….**

➔ **Temps de calcul < 1 ms (objectifs < 10 µs)**

- Modèles disponibles dans THINK:
  - **Débruitage par CNN**
  - Désempilement par MLP, RNN, CNN
  - **Debruitage et deconvolution par 1DCNN**
  - Détection d'évenements rare (anomalie)

L'apprentissage profond embarqué sur AMD-Xilinx – Etat des lieux –



Principe de THINKML: Maîtriser le Gateware pour maîtriser les ressources du FPGA

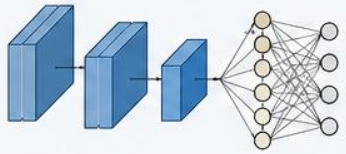
THINKML

De vos modèles Keras à l'IP FPGA optimisée

Inférence rapide. Précision maîtrisée.

1. MODÈLE KERAS
Python

```
from tensorflow import keras
model = keras.Sequential([
   keras.layers.Conv2D(32, (3,3),
       activation='relu', input_shape=(28,28,1))
   keras.layers.MaxPooling2D((2,2))
   ...
   keras.layers.Dense(10, activation='softmax')
])
model.compile(...)
```



2. THINKML
Conversion

THINKML

- Analyse du modèle Keras
- Extraction des couches et paramètres
- Quantification & optimisation
- Génération VHDL paramétrable

3. GÉNÉRATION VHDL
IP paramétrable

Entité VHDL THINKML_IP

Paramètres configurables

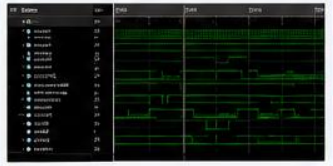
Nombre de DSP — 64

Filtres CNN en parallèle — 8

Architecture pipeline & parallélisme optimisée pour FPGA

4. SIMULATION
cocoTb

- Testbench automatique
- Stimuli & réponses
- Vérification fonctionnelle
- Comparaison avec Keras



5. IMPLÉMENTATION
FPGA Xilinx Zynq

VIVADO HLx Editions

XILINX ZYNQ Processing System

6. ENVIRONNEMENT DE TEST AUTOUR DE L'IP – PLATINE ZYNQ

FPGA (PL)

THINKML_IP

CNN / DNN Inference Engine

DSP Configurés | Filtres en parallèle

AXI Interconnect

ARM Cortex-A9 (PS)

Serveur de commandes Unicode

- Interface série (UART)
- Commandes Unicode
- Chargement des données
- Lancement de l'inférence
- Récupération des résultats

Interfaces

UART
DDR
GPIO
...

7. MESURES & VALIDATION

Temps de latence

Mesure du temps d'inférence sur la cible FPGA

Précision des résultats

Comparaison avec le modèle Keras (référence)

Accélération matérielle grâce au parallélisme

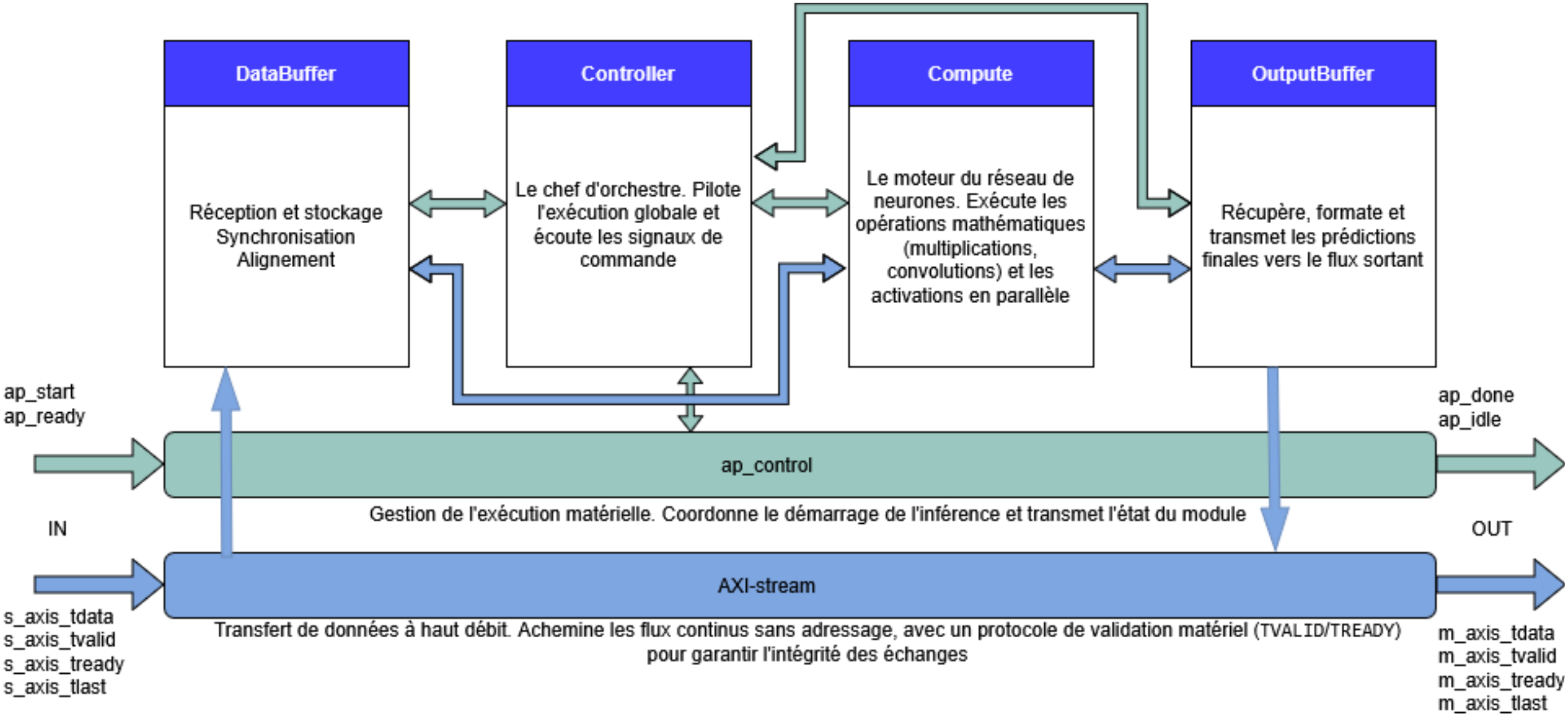
Ressources maîtrisées (DSP, BRAM, LUT)

Haute précision préservée

Flux automatisé de bout en bout

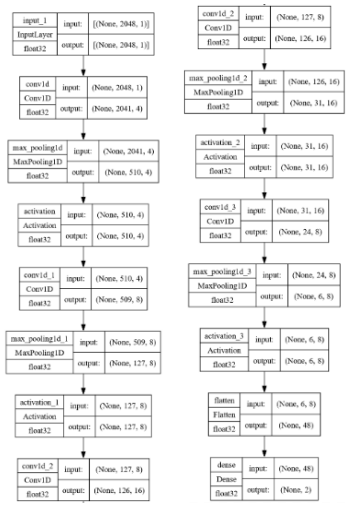
THINKML : votre passerelle de l'IA à l'embarqué
Performances, flexibilité et fiabilité sur FPGA.

THINKML: L'entité VHDL de base

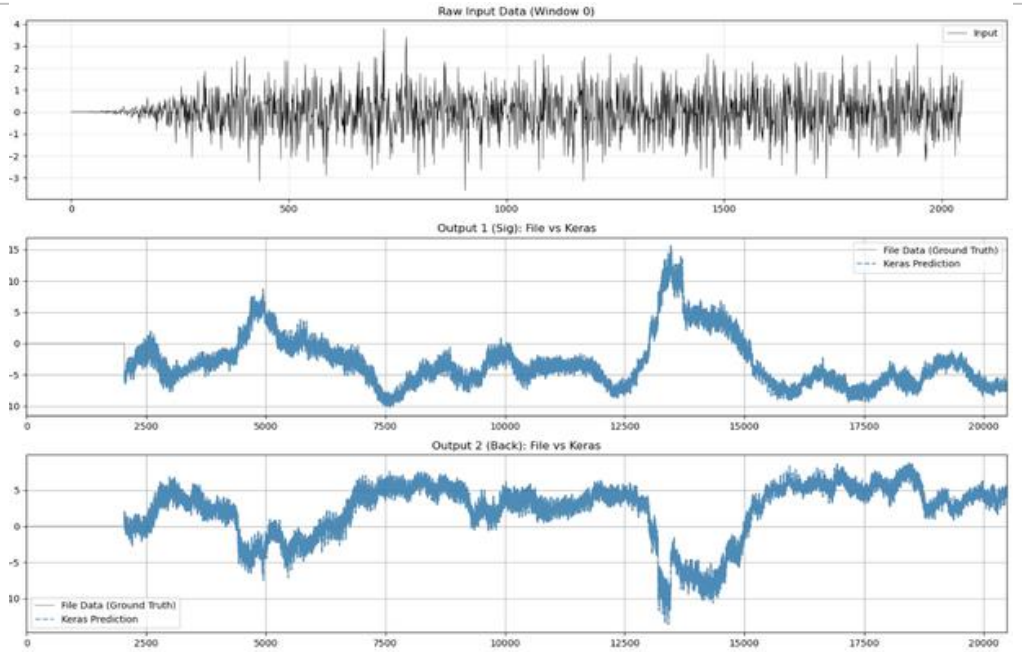


Comparaison des résultats HLS4ML versus THINKML

1- Modèle CNN de débruitage des ondes gravitationnelles



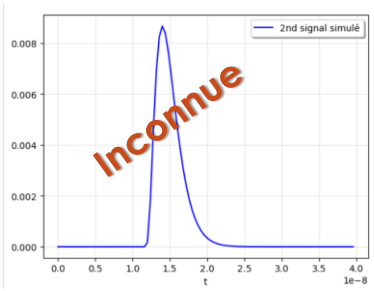
- 4 couches de convolution
- 1 couche dense
- 37094 paramètres



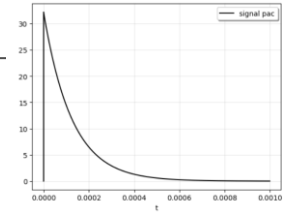
➔ **Input**

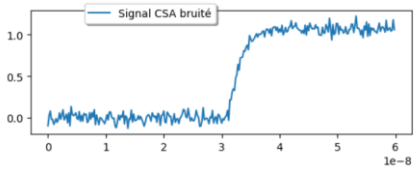
➔ **Output**

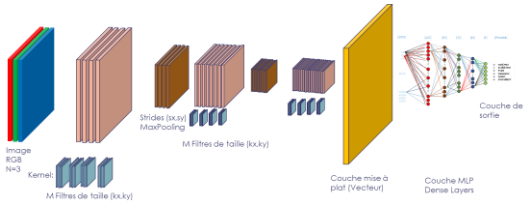
2- Modèle 1D CNN dilaté de débruitage et déconvolution d'une chaîne électronique avec préampli de charge

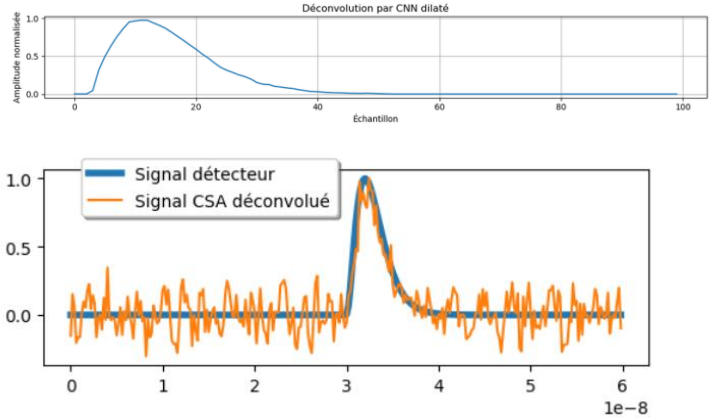


PAC









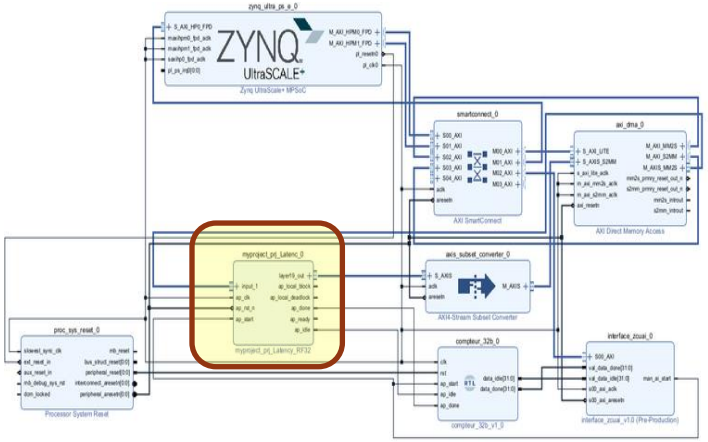
➔ Problème mal posé au sens de Hadamard (solutions multiples)

➔ Travail en cours

Exemple modèle de débruitage des Ondes gravitationnelles – HLS4ML -

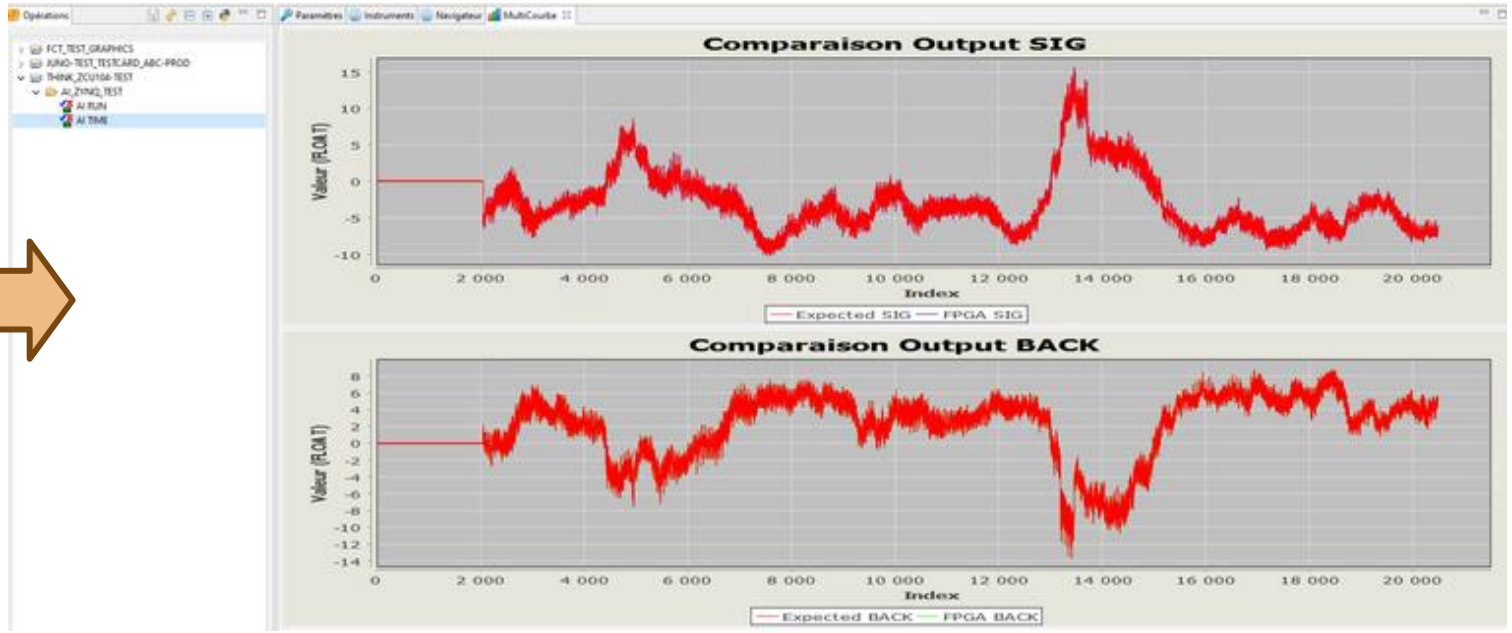
hls 4 ml

(cible ZCU104)



Client DAVIS

(Data Acquisition Verification Integration system)



1- **HLS4ML optimise lui-même la description du reseau en HLS**

2- **HLS synthétise le code VHDL de l'IP avec peu d'influence du concepteur**

3- **Les ressources utilisées et le temps d'inférence sont constatés après coup**

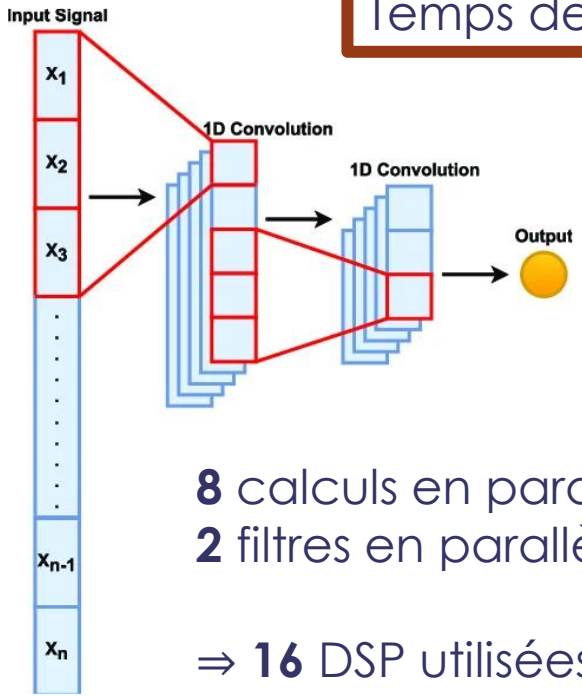
4- **Le concepteur ne maîtrise pas le code VHDL créé**

	Utilisation des Ressources				Performance du modèle	
RF= Reuse Factor Optimisation en Ressource ou latence	DSP	LUT	FF	BRAM	Nombre de cycles	Temps inférence
Resource_RF_32	48	70374	26599	165	N/A	N/A
Latency_RF_32	46	69728	33402	137	88248	265.01 us
Resource_RF_2	736	36650	63653	466	39066	117.32 us
Latency_RF_2	736	67301	20274	137	32896	98.79 us
Ressources maximales disponibles	1728	230400	460800	312 (*)		

Exemple modèle de débruitage des Ondes gravitationnelles – THINKML -

```
Couche                  | Arch    | (K, C_IN, NB_FILTRE)      | DSP Réels  | PARALLEL_CH | DSP/CH
-------------------------------------------------------------------------------------------------
conv1d                  | v3      | (8, 1, 4)                 | 16         | 2           | 8
max_pooling1d           | common  | (_, 4, 4)                 | 0          | -           | -
activation              | common  | (_, 4, 4)                 | 0          | -           | -
conv1d_1                | v3      | (2, 4, 8)                 | 32         | 4           | 8
max_pooling1d_1         | common  | (_, 8, 8)                 | 0          | -           | -
activation_1            | common  | (_, 8, 8)                 | 0          | -           | -
conv1d_2                | v3      | (2, 8, 16)                | 32         | 2           | 16
max_pooling1d_2         | common  | (_, 16, 16)               | 0          | -           | -
activation_2            | common  | (_, 16, 16)               | 0          | -           | -
conv1d_3                | v3      | (8, 16, 8)                | 128        | 2           | 64
max_pooling1d_3         | common  | (_, 8, 8)                 | 0          | -           | -
activation_3            | common  | (_, 8, 8)                 | 0          | -           | -
dense                   | v3      | (6, 8, 2)                 | 64         | 2           | 32
-------------------------------------------------------------------------------------------------
>> TOTAL DSP CONSOMMÉS : 272 / 1728
```

THINKML vs HLS4ML:
Nbre de DSP divisé par 3
Temps de latence 100x plus rapide



1- **Contrôle total du nombre de DSP**
2- **Optimisation des latences par couche**
3- Aucun contrôle des LUT et FF (Optimisation Vivado)

8 calculs en parallèle sur les $K*C_{in\ à\ faire}$
2 filtres en parallèle sur les NB_FILTRE à faire

⇒ **16** DSP utilisées pour la première

Optimisation nbre DSP et calcul paralleles	DSP	LUT	FF	BRAM	Nombre de cycles	Temps inférence
Modele OG	272	30168	24126	11,5	115	1,15µs
Ressources max. disponibles	**1728**	**230400**	**460800**	**312**		

Exemple de déconvolution des signaux – THINKML –

Layer (type)	Output Shape	Param #
signal_input (InputLayer)	(None, 100, 1)	0
dilated_1_pruned (Conv1D)	(None, 100, 9)	36
dilated_2_pruned (Conv1D)	(None, 100, 64)	1,792
dilated_4_pruned (Conv1D)	(None, 100, 64)	12,352
dilated_8_pruned (Conv1D)	(None, 100, 64)	12,352
dilated_16_pruned (Conv1D)	(None, 100, 64)	12,352
dilated_32_pruned (Conv1D)	(None, 100, 64)	12,352
final_output (Conv1D)	(None, 100, 1)	65

Total params: 51,301 (200.39 KB)

Trainable params: 51,301 (200.39 KB)

hls4ml

(cible ZCU104)

Modèle de base:

- 6 couches de CNN
- 0 couche dense
- 51301 paramètres

Echec de l'implémentation HLS
Indétermination de la sortie
➔ Absence MLP

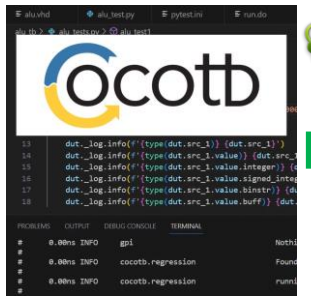
Simplification du modèle (test)

Layer (type)	Output Shape	Param #
signal_input (InputLayer)	[(None, 100, 1)]	0
dilated_1 (Conv1D)	(None, 100, 16)	64
dilated_2 (Conv1D)	(None, 100, 16)	784
dilated_4 (Conv1D)	(None, 100, 16)	784
dilated_8 (Conv1D)	(None, 100, 16)	784
dilated_16 (Conv1D)	(None, 100, 16)	784
final_output (Conv1D)	(None, 100, 1)	17

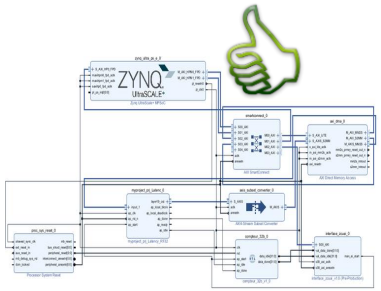
Total params: 3217 (12.57 KB)
Trainable params: 3217 (12.57 KB)
Non-trainable params: 0 (0.00 Byte)

THINKML

Modèle VHDL



Simulation avec cocotb

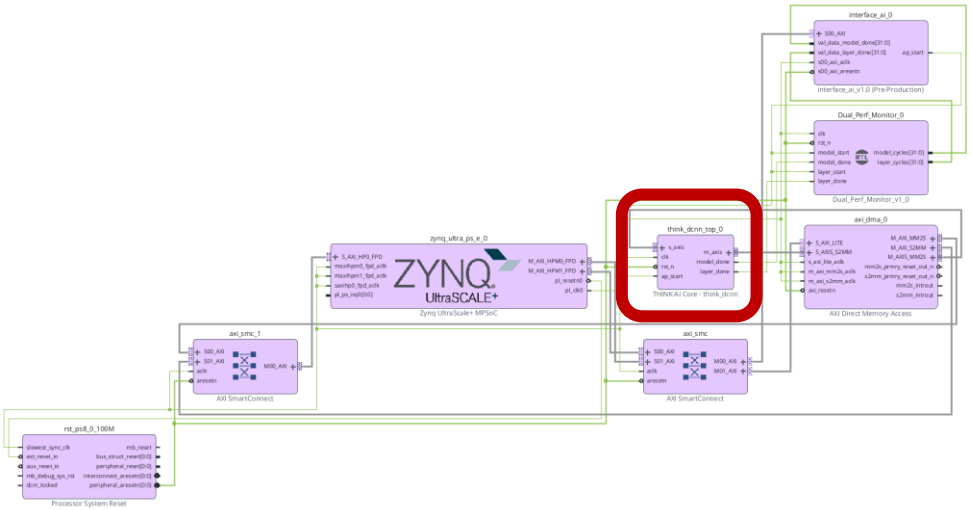


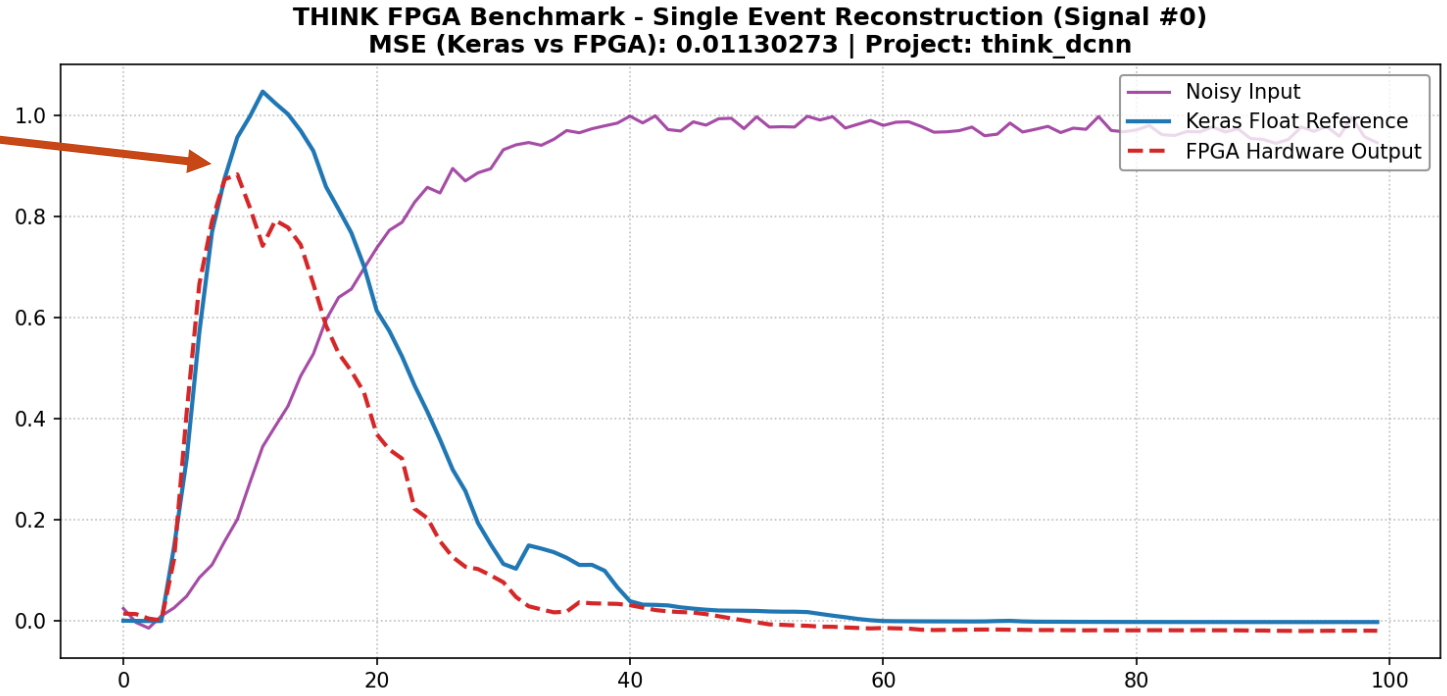
Projet Vivado OK

➔ Maîtrise du nombre de DSP
➔ Maîtrise du nombre de calcul en parallèle
➔ Maitrise des buffers d'entrée et du modèle
➔ Non maîtrise de l'utilisation des LUT
➔ Peut bloquer la synthèse du design

Exemple de déconvolution des signaux – THINKML –

Erreur d'amplitude :
Différence de Gain lié à la quantization **<16,3>** ?

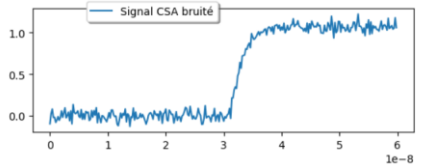




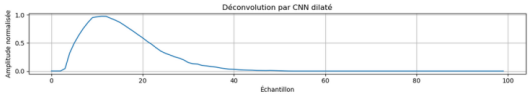
Rmq: 2x Nbre DSP / OG ➔ 2 x plus lent.

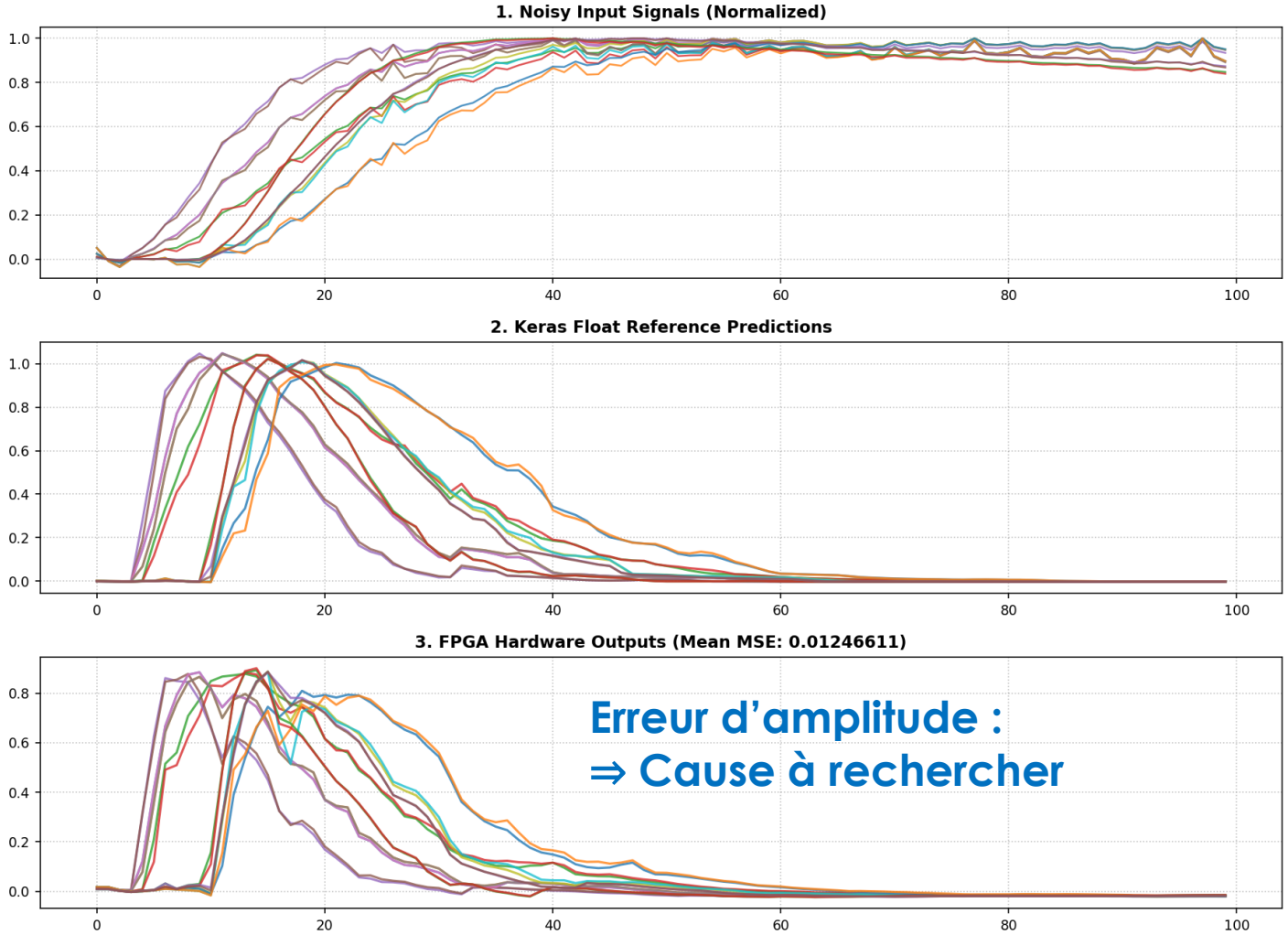
Optimisation nbre DSP et calcul paralleles	DSP	LUT	FF	BRAM	Nombre de cycles	Temps inférence
Modele OG	423	161387	49621	18,5	213	2,13µs
Ressources max. disponibles	**1728**	**230400**	**460800**	**312**		

Exemple de déconvolution des signaux – THINKML –



VS





🡸 **Signal sortie PAC**

🡸 **Signal sortie Modèle reference Keras (Modèle python TF)**

🡸 **Signal sortie FPGA (IP VHDL)**

Conclusion: TNS avec des modèles d'IA validé

- HLS4ML OK sur FPGA puissant
  - **Difficile de maîtriser la latence** (analyse à posteriori)
  - Maîtrise de la quantisation
  - **Echec de l'implémentation possible**

- THINKML OK pour tout FPGA (AMD)
  - Génération d'un modèle **VHDL** générique 1D CNN & MLP
  - Maîtrise du temps de calcul en fonction des ressources
  - Simulation avant synthèse FPGA (**cocotb**)
  - Banc de test Matériel sur FPGA (client/serveur)
  - ***Non maîtrise des estimations des LUT (taille des données d'entrées)***
    - ➔ ***Comprendre***
    - ➔ ***Amélioration à étudier***
  - ***Etude 2D CNN à développer***
  - ***Consolidation des modèles (données réelles)***