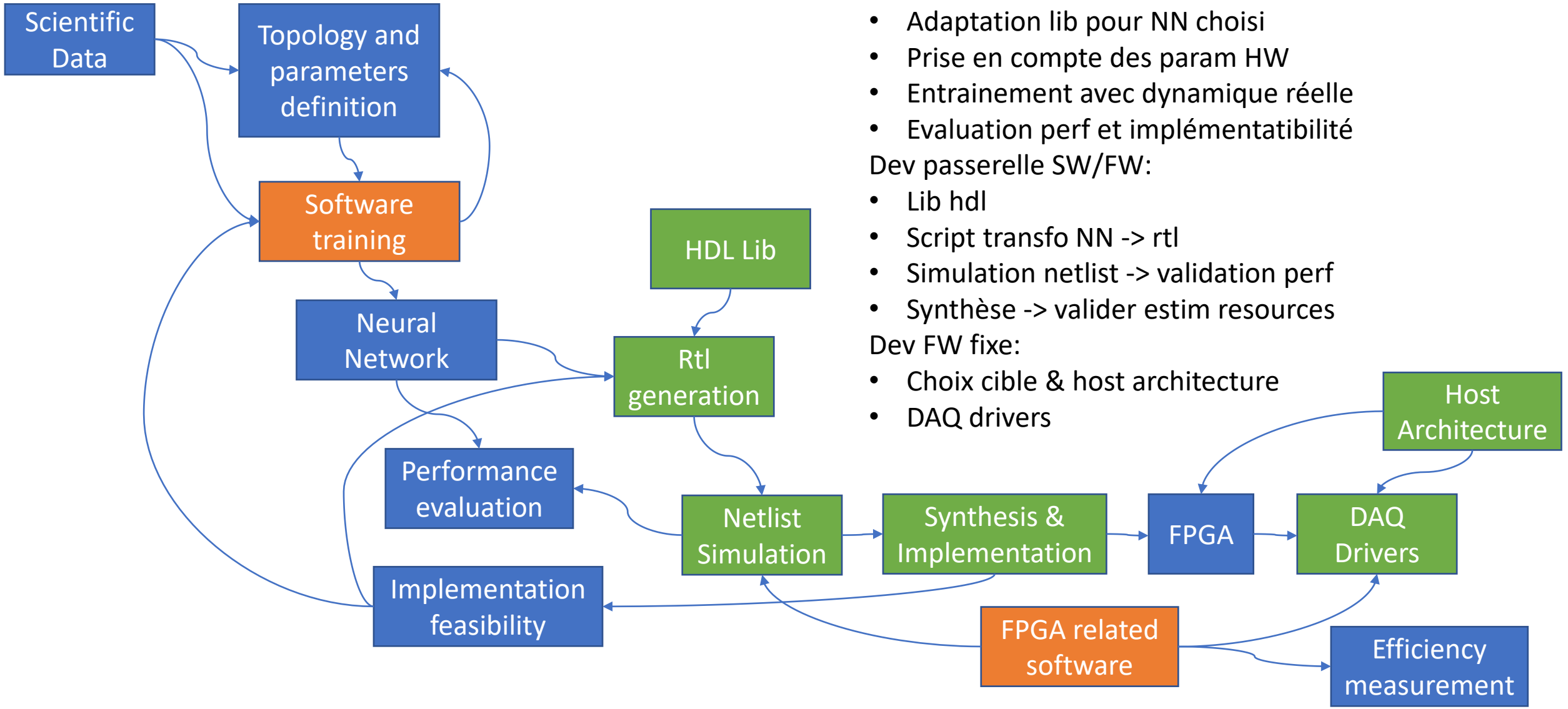
Démonstrateur pour le portage HW automatisé



Dev SW:

- Adaptation lib pour NN choisi
- Prise en compte des param HW
- Entrainement avec dynamique réelle
- Evaluation perf et implémentatibilité

Dev passerelle SW/FW:

- Lib hdl
- Script transfo NN -> rtl
- Simulation netlist -> validation perf
- Synthèse -> valider estim resources

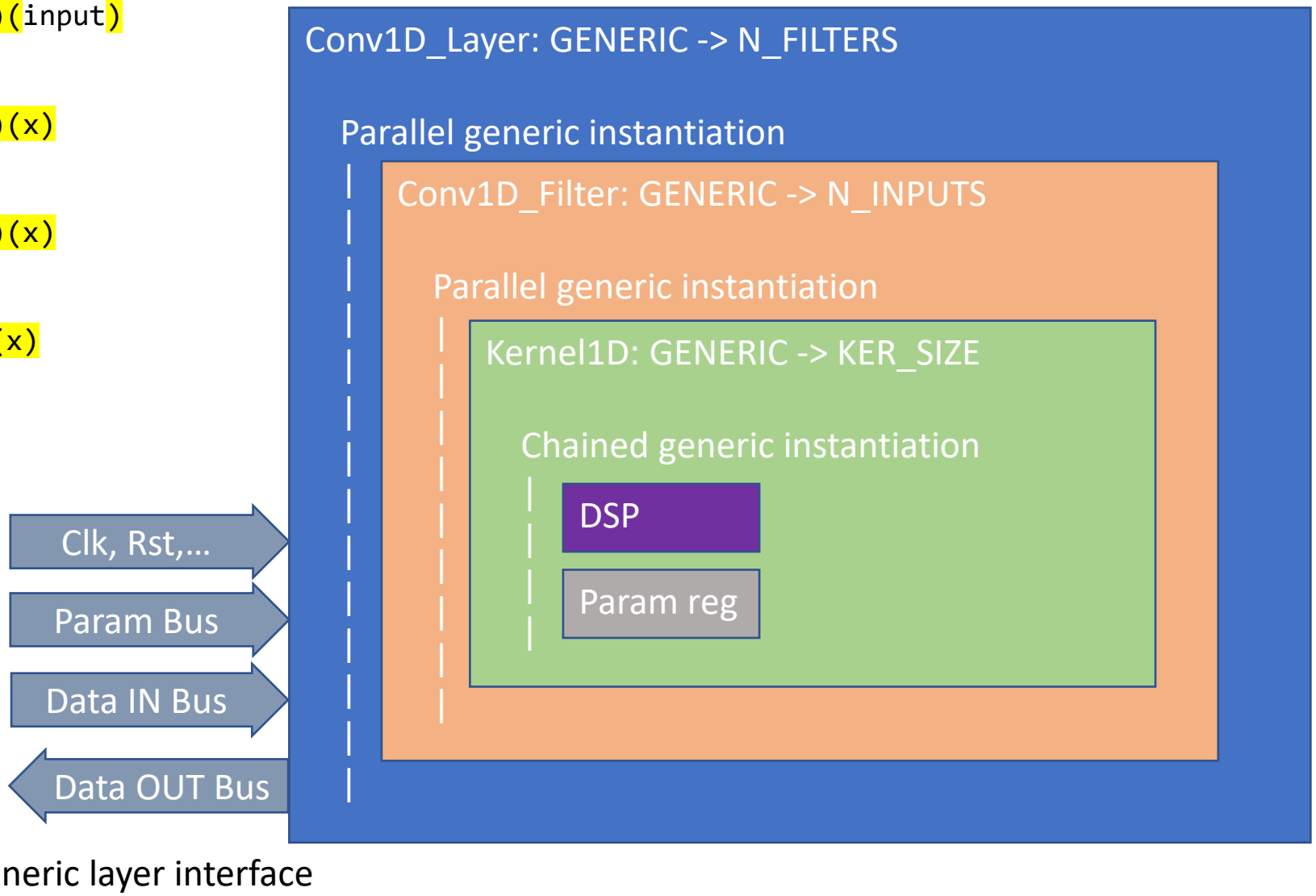
Dev FW fixe:

- Choix cible & host architecture
- DAQ drivers

Library HDL - description des composants nécessaires

TensorFlow description

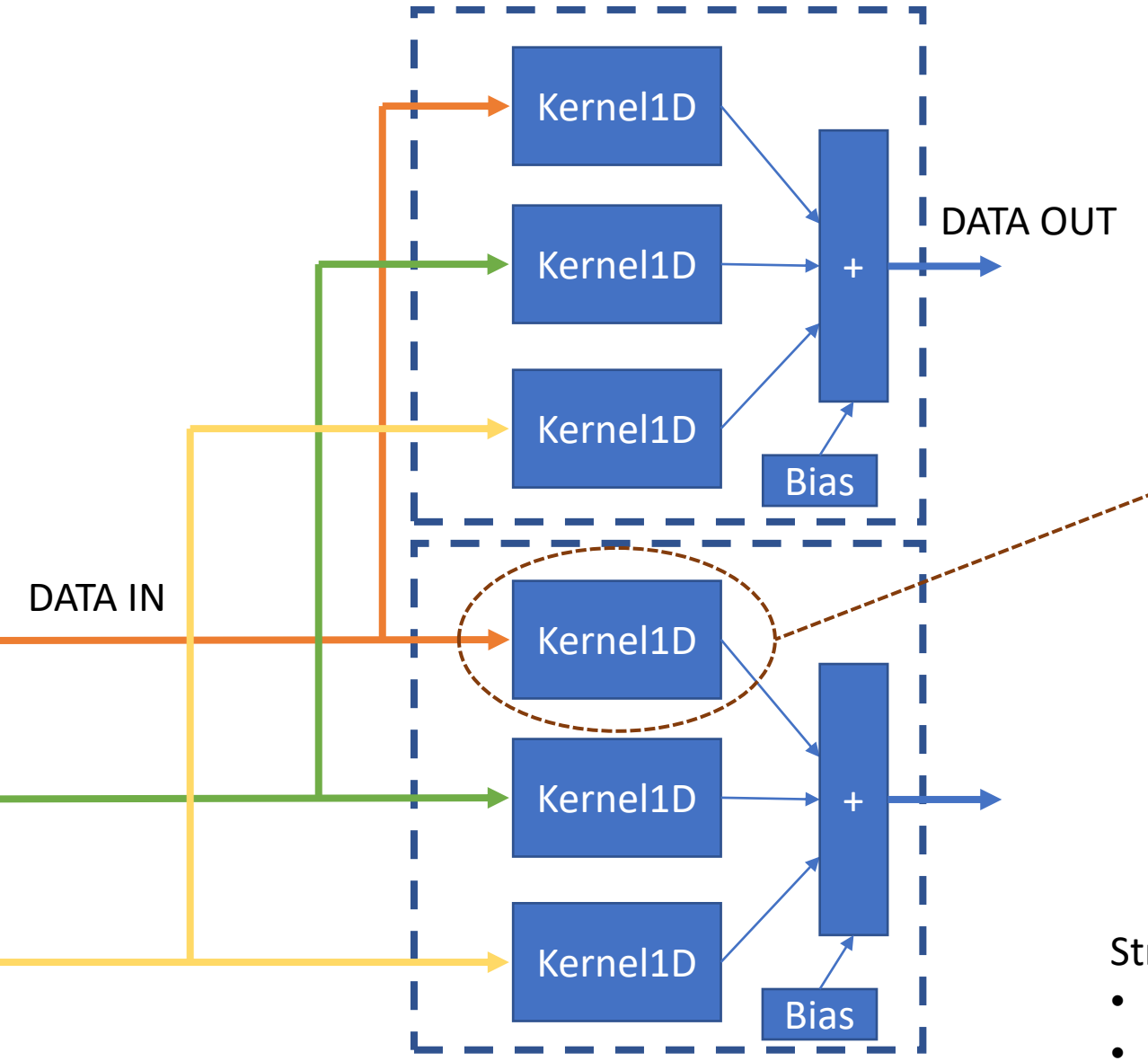
```
x=layers.Conv1D(filters=16, kernel_size=8)(input)
x=layers.MaxPool1D(pool_size=4)(x)
x=layers.Activation(activation='relu')(x)
x=layers.Conv1D(filters=32, kernel_size=2)(x)
x=layers.MaxPool1D(pool_size=4)(x)
x=layers.Activation(activation='relu')(x)
x=layers.Conv1D(filters=16, kernel_size=2)(x)
x=layers.MaxPool1D(pool_size=4)(x)
x=layers.Activation(activation='relu')(x)
x=layers.Conv1D(filters=8, kernel_size=8)(x)
x=layers.MaxPool1D(pool_size=4)(x)
x=layers.Activation(activation='relu')(x)
x=layers.Flatten()(x)
output=layers.Dense(2)(x)
```



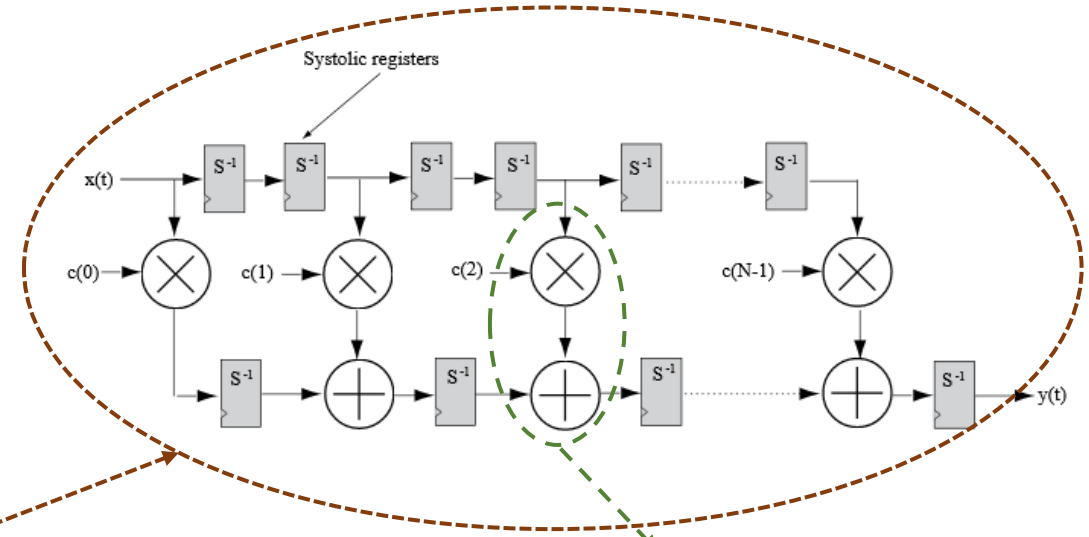
Generic layer interface

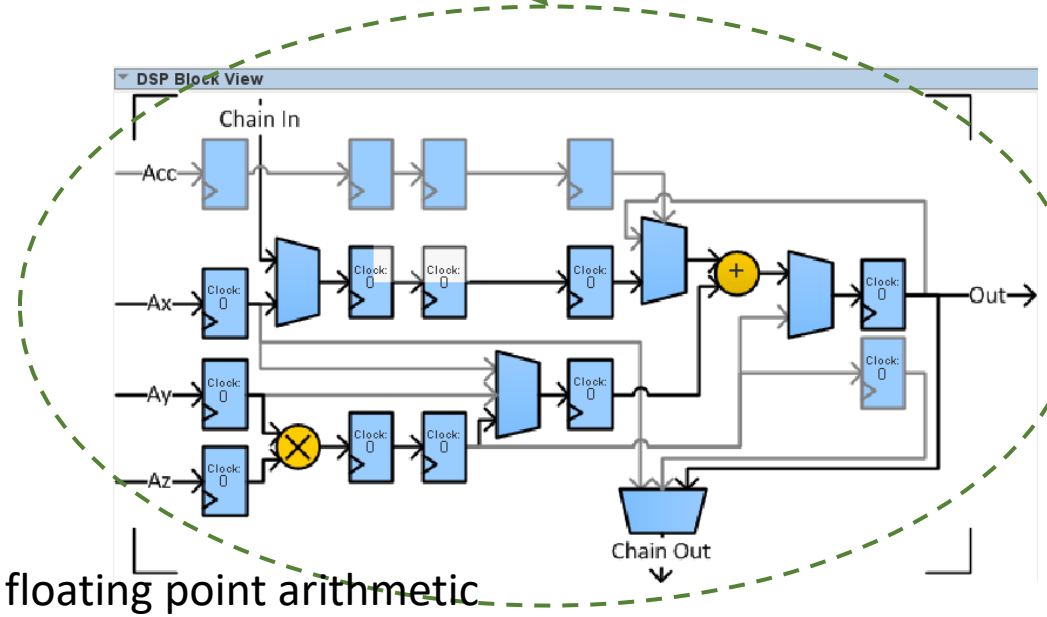
Library HDL - description des composants nécessaires

Conv1D_Layer (N_INPUTS => 3, N_FILTERS => 2)



Kernel 1D module = FIR filter

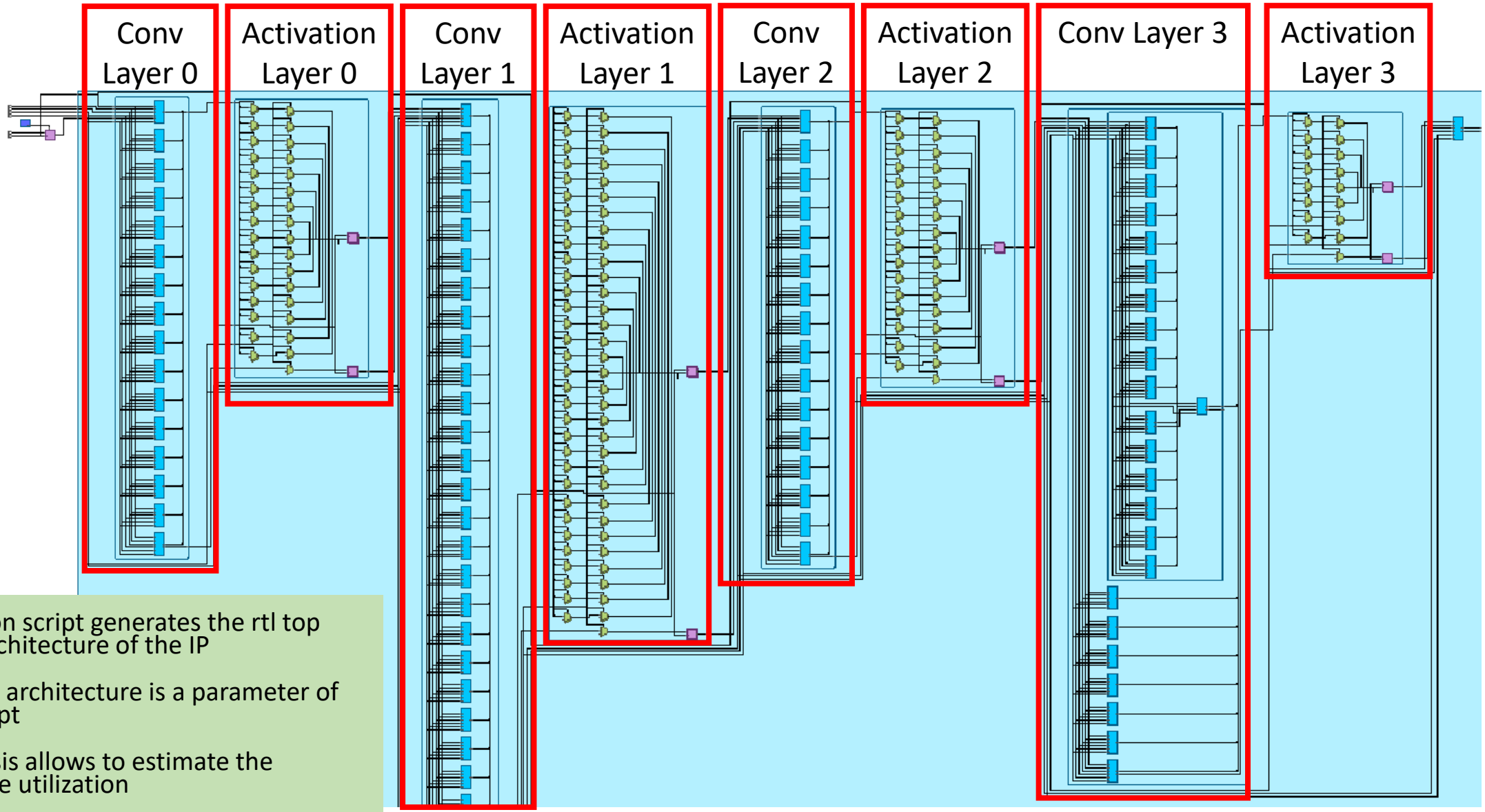




Stratix 10 DSP

- Native 32b floating point arithmetic
- Up to 4000 units available

RTL Generation script



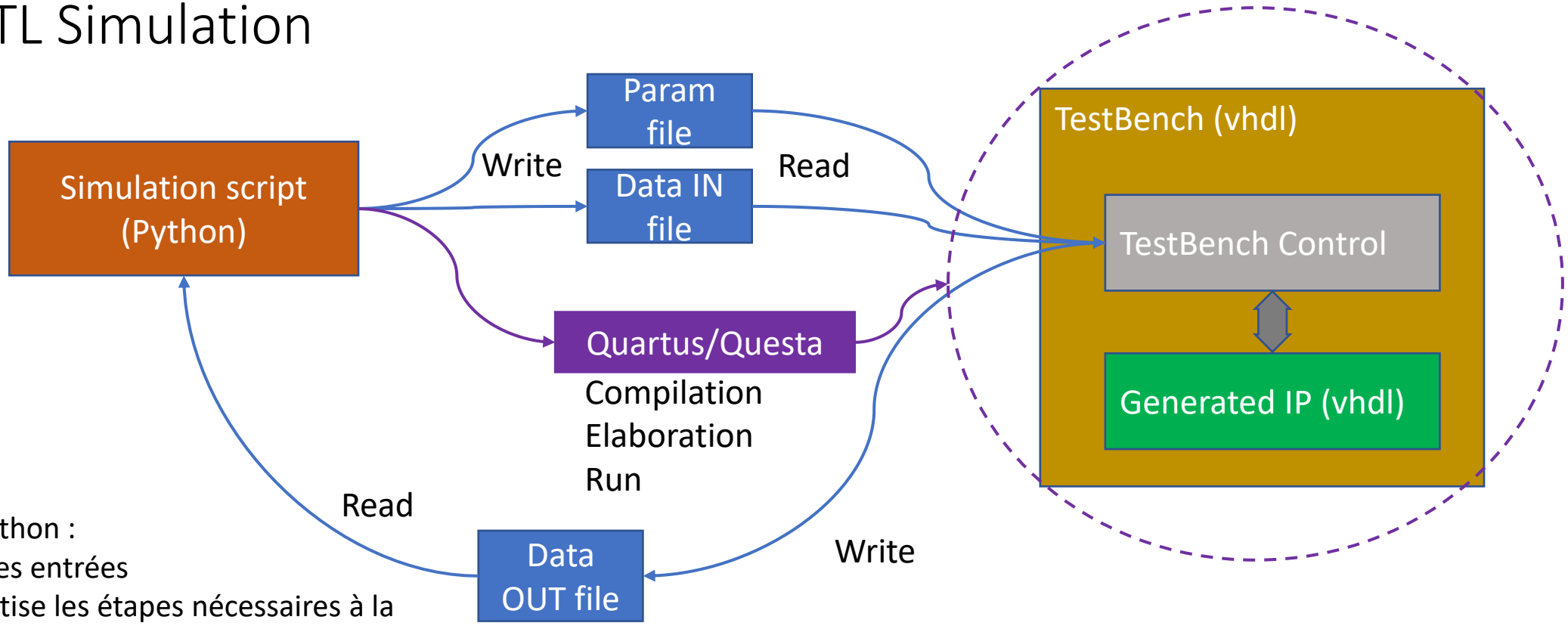
A python script generates the rtl top level architecture of the IP

Desired architecture is a parameter of the script

Synthesis allows to estimate the resource utilization

Fully unrolled architecture (no reusability) -> only small design can fit in FPGA

RTL Simulation

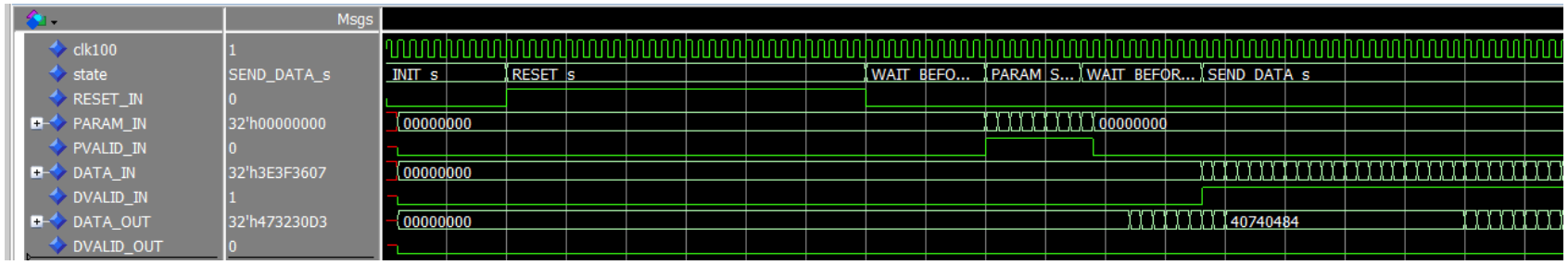


Le script Python :

- Fourni les entrées
- Automatise les étapes nécessaires à la simulation
- Récupère les résultats et les analyse (comparaison avec calcul soft)

Le TestBench :

- Génère l'horloge
- Instancie l'IP générée
- Séquence le test
- Fait des accès aux fichiers

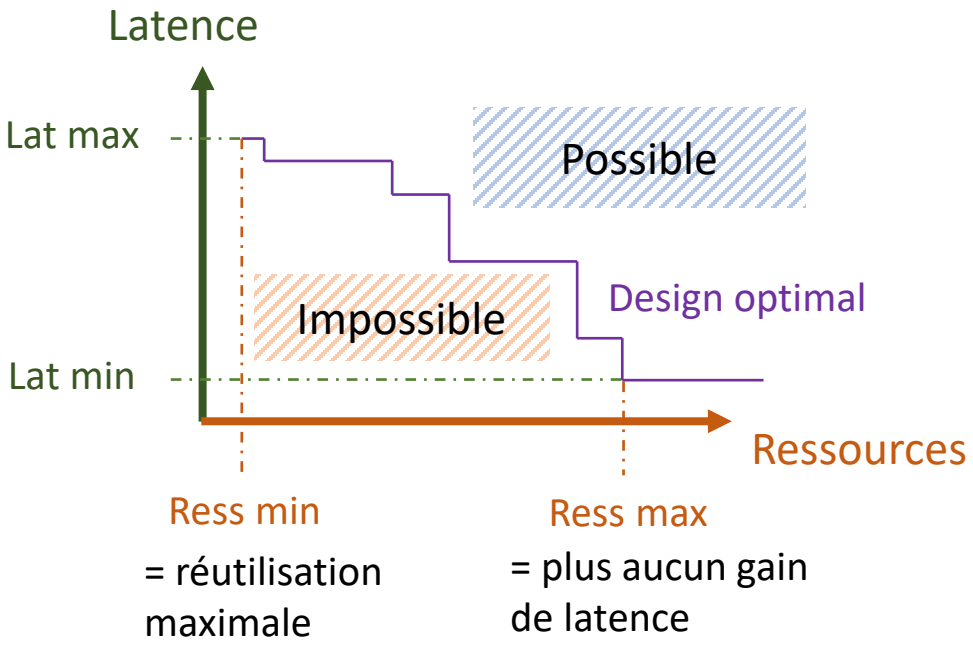


How to fit the system in a FPGA

TradeOff entre différentes grandeurs :

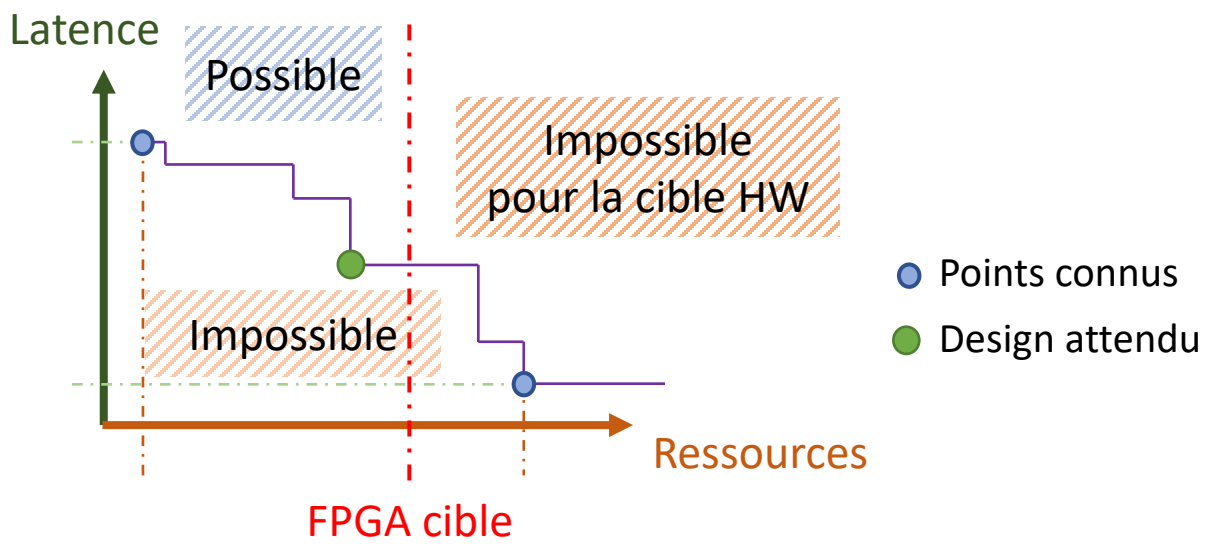
- Ressources disponibles (DSP, mémoire, routage …) ← Dépends du FPGA et du système hôte
- Latence
- Précision
- Consommation électrique
- …

Contraintes extérieures



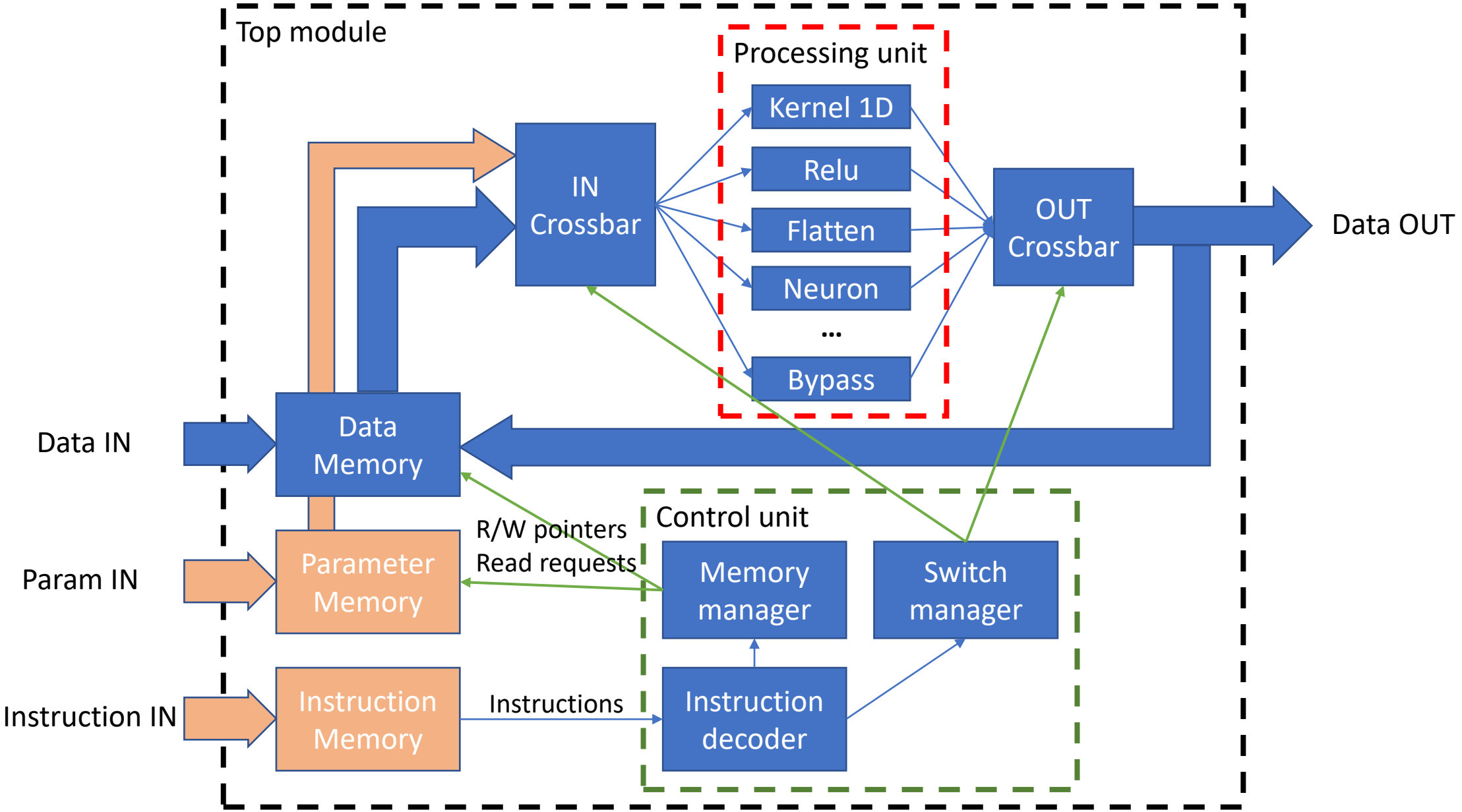
= réutilisation maximale

= plus aucun gain de latence

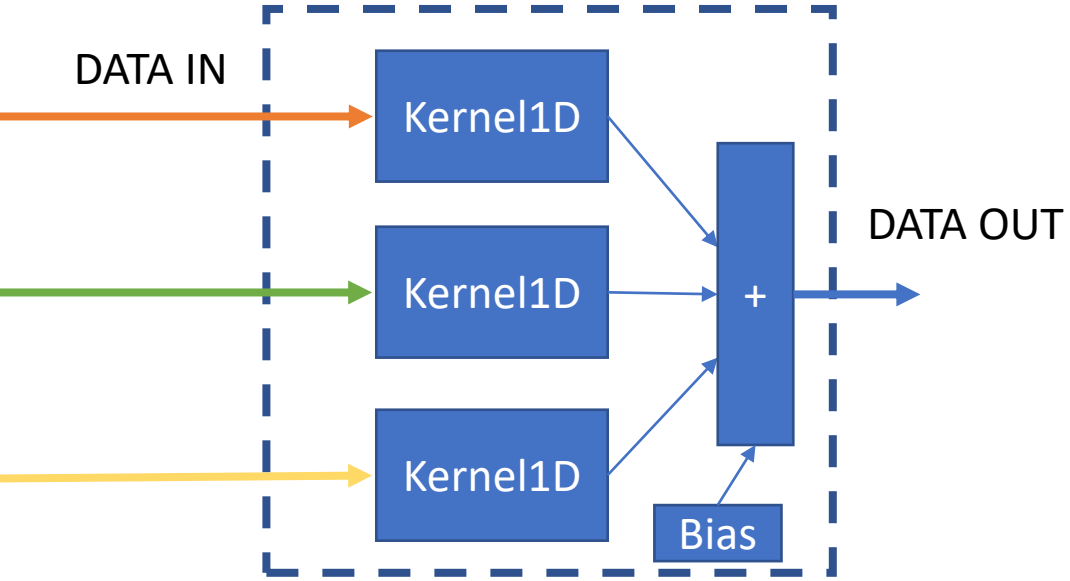


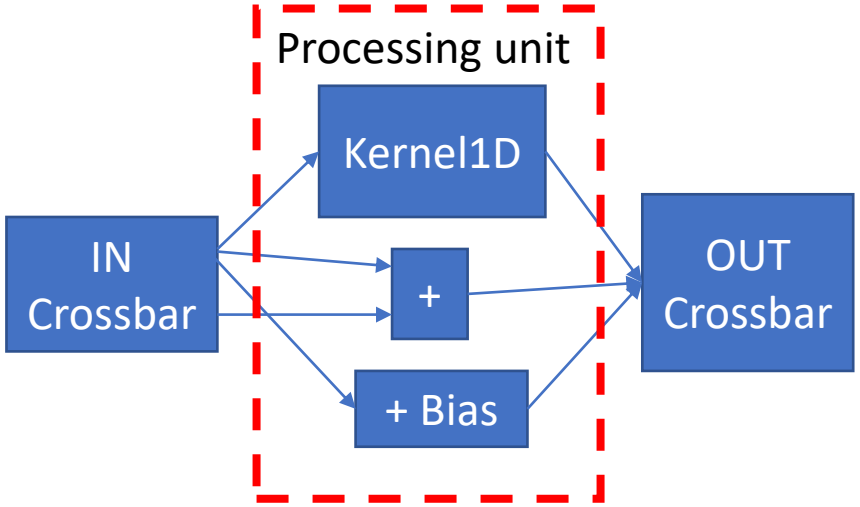
Approche choisie pour converger vers le design cible : Partir du design minimal (latence max) et en itérer en dupliquant le module de calcul permettant le meilleur gain en latence (valeur facile à estimer).

Minimal design, highest resources reusability



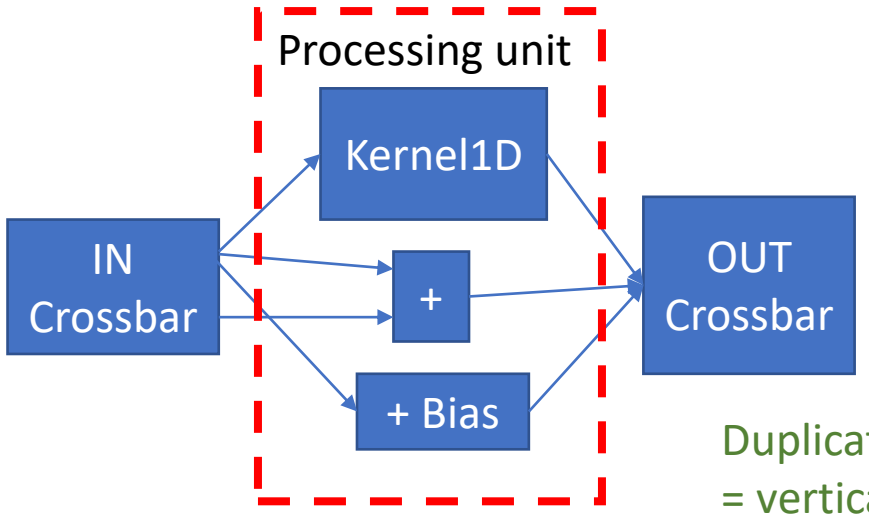
Exemple d'instructions



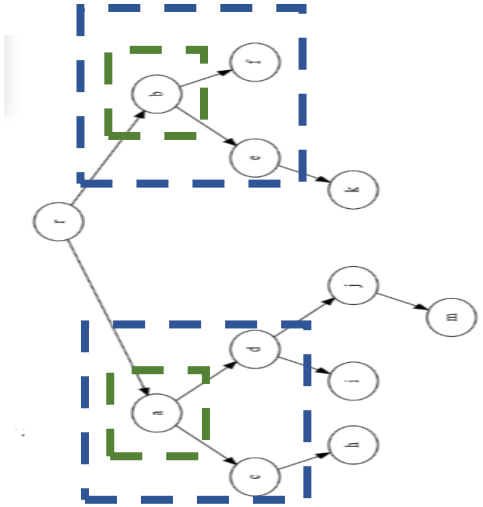



```
Set CROSSBAR MODE -> KERNEL 1D (1 Dataflow)
Set PRP = K0 param Base addr
Read Param 8
Set DWP = res0 Base addr
Set DRP = In0 data Base addr
Read Data 2048
Set PRP = K1 param Base addr
Read Param 8
Set DWP = res1 Base addr
Set DRP = In1 data Base addr
Read Data 2048
Set PRP = K2 param Base addr
Read Param 8
Set DWP = res2 Base addr
Set DRP = In2 data Base addr
Read Data 2048

Set CROSSBAR MODE -> Reduction Adder (2 Dataflows)
Set DRP0 = res0 Base addr
Set DRP1 = res1 Base addr
Set DWP = res0+1 Base addr
Read Data 2048
Set DRP0 = res0+1 Base addr
Set DRP1 = res2 Base addr
Set DWP = res0+1+2 Base addr
Read Data 2048

Set CROSSBAR MODE -> Constant Adder (1 Dataflow)
Set PRP = Bias0 param Base addr
Read Param 1
Set DRP = res0+1+2 Base addr
Set DWP = res0+1+2 Base addr
Read Data 2048
```

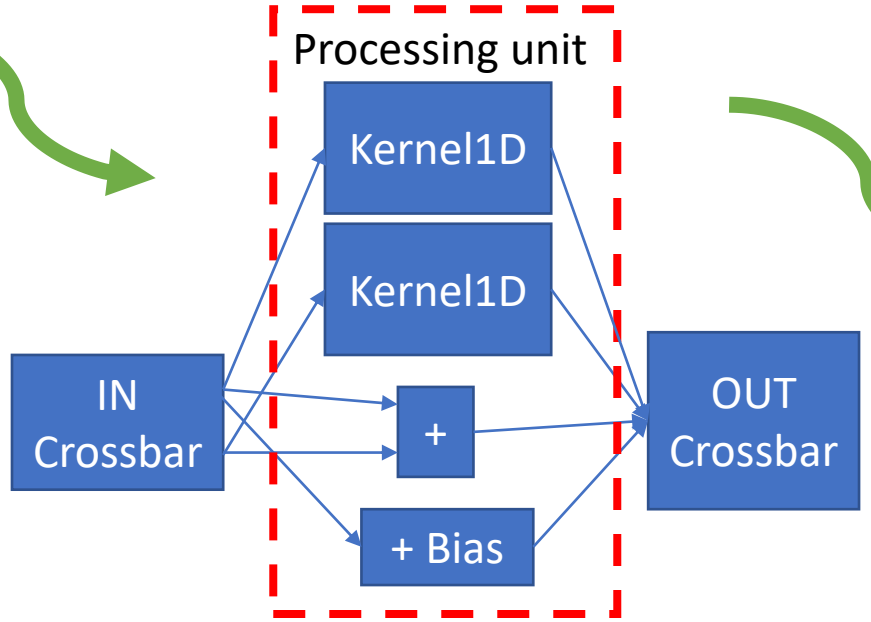
Evolution par itérations successives



- Guidée par une recherche de « patterns » sur l'arbre de calcul
- Les instructions sont regénérées pour chaque nouvelle architecture



Duplication de modules les plus utilisés
= verticale



Prolongement des blocs
= horizontal