MULTIPROCESSOR SYSTEM ON CHIP :

Robot Husky

Groupe 1

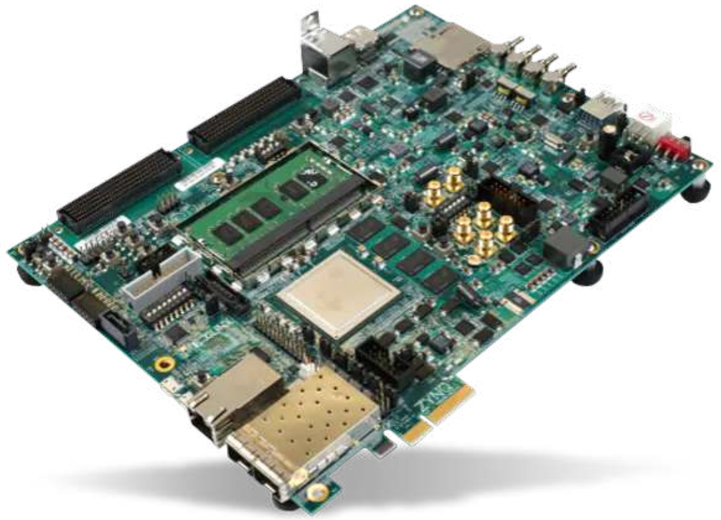
Plan

Introduction

Objectif

Dans ce projet de ROB307, on cherche à étudier et réaliser un MPSOC (Multiprocesors System on Chip) sur le circuit configurable FPGA XC7Z020 disponible sur Zedboard.

Notre but est d'exécuter plusieurs fonctions en même temps sur un robot Husky tout en évaluant l'aspect énergétique.



Network on chip (NOC)

Conception du NOC

Objectif :

- Construire un design de base qui permet d'effectuer des traitements parallèles sur un ou plusieurs processeurs.

Principe :

- Le protocole choisi pour la communication : AXI4
- Le cœur : AXI Crossbar

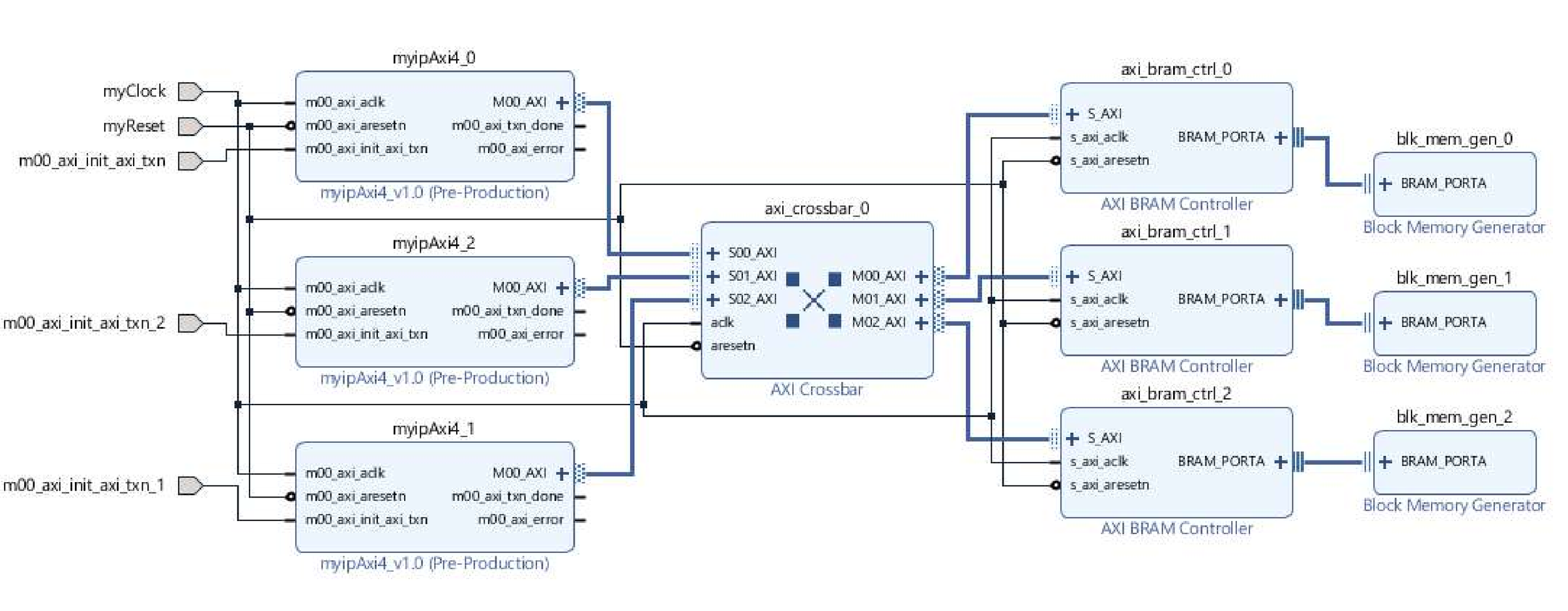
Les modes de connectivité :

- Full Crossbar mode
- Shared Access Mode

La génération des signaux :

- Signaux décalés
- Signaux générés aux même instant

Conception du NOC 3x3



Simulation du NOC 3x3

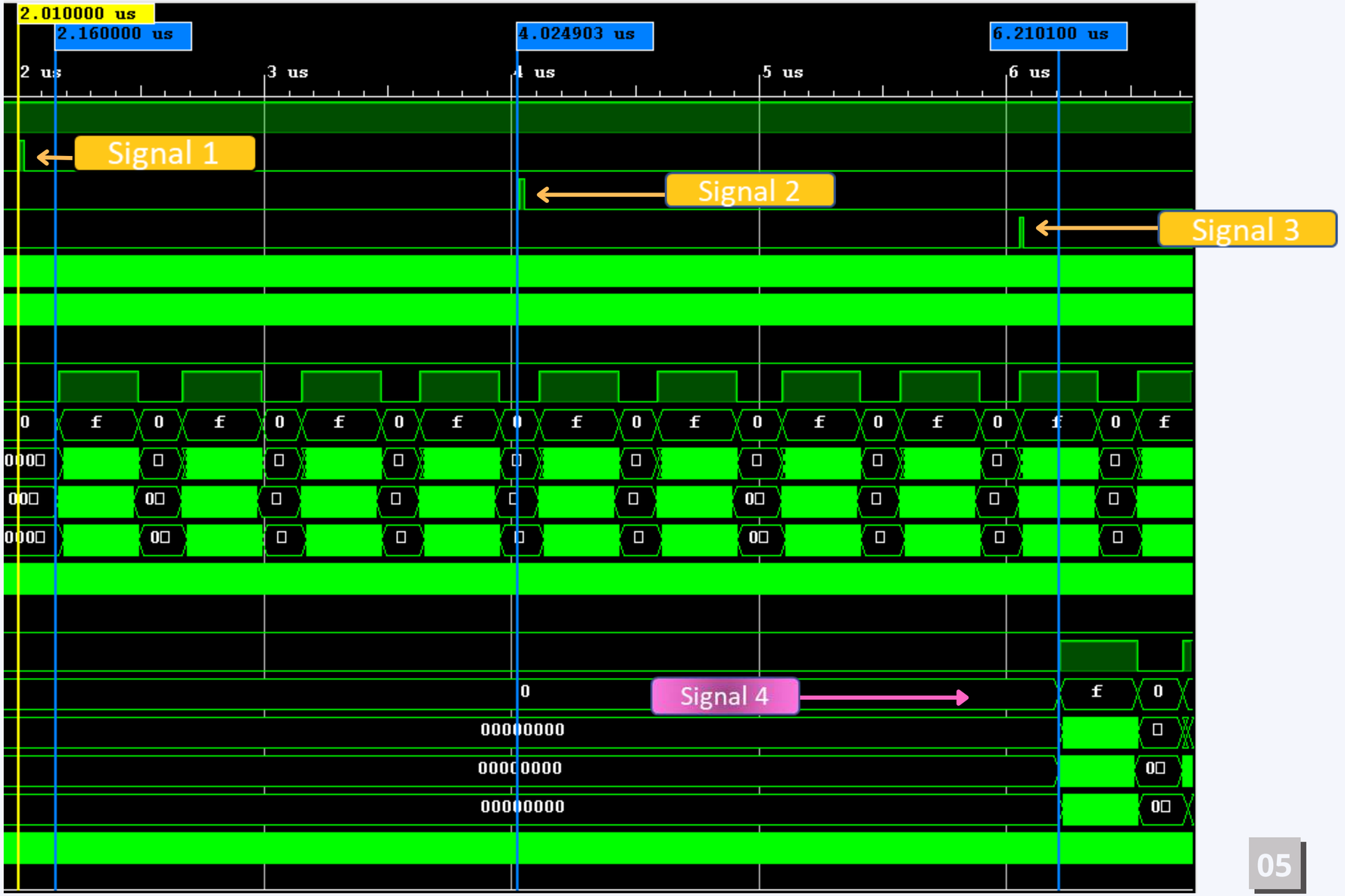
Deux modes de l’axi-crossbar :

- Crossbar : Chemins de traverse parallèles pour les canaux de données d'écriture et de données de lecture
- Shared-access : Données d'écriture partagées, données de lecture partagées et chemins d'adresse uniques partagés.

Deux types de signaux :

- Signaux décalés : les signaux de déclanchement sont décaler temporairement
- Signaux non décalés : les signaux de déclanchement de l'écriture sont lancer simultanément

Simulation du NOC 3x3



Signal 1 : signal d'initialisation de l'API 1

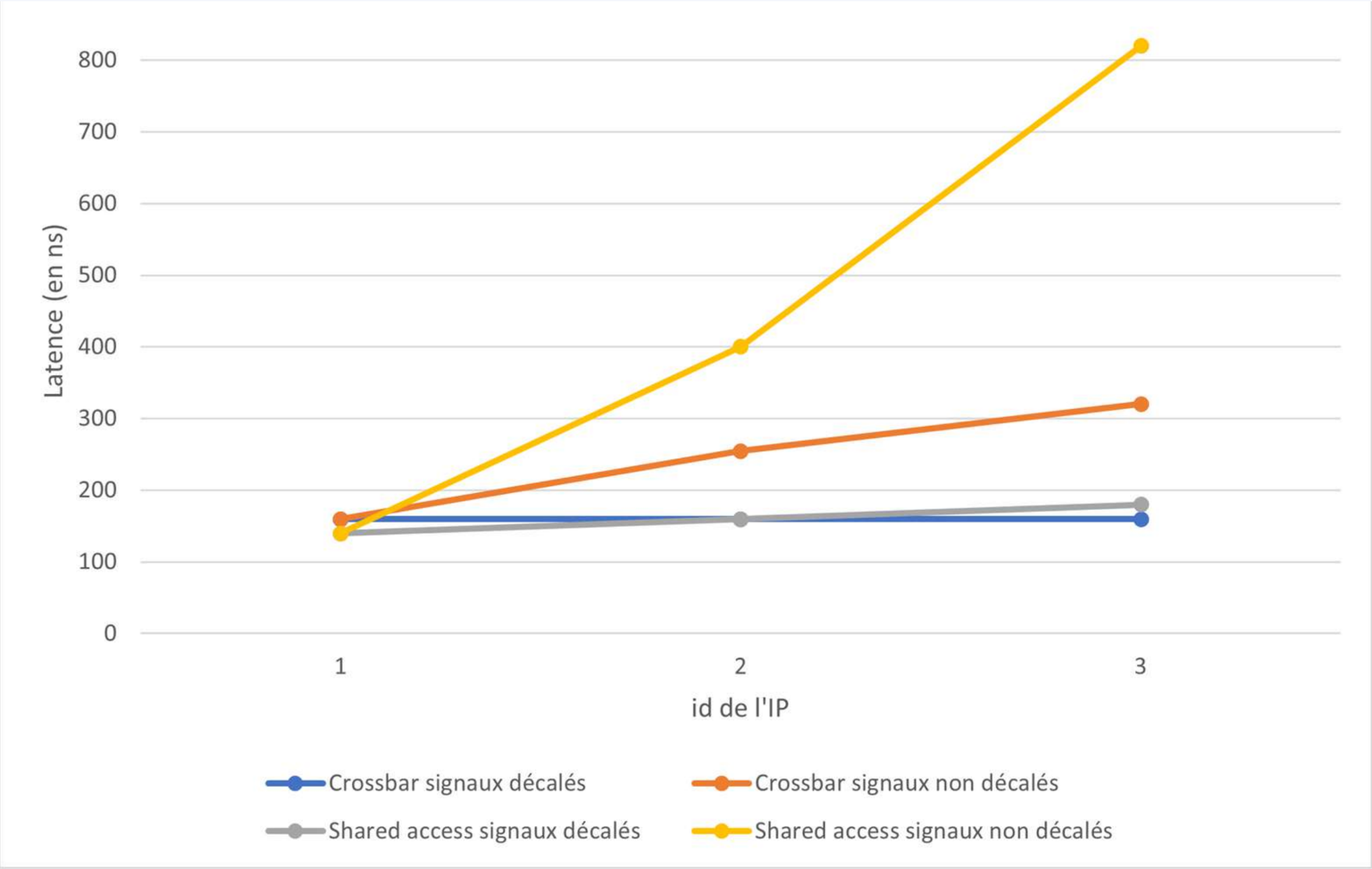
Signal 2 : signal d'initialisation de l'API 2

Signal 3 : signal d'initialisation de l'API 3

Signal 4 : signal de l'écriture sur la mémoire des données de l'API 1

Cas ou les 3 signaux sont décalés

Latence :

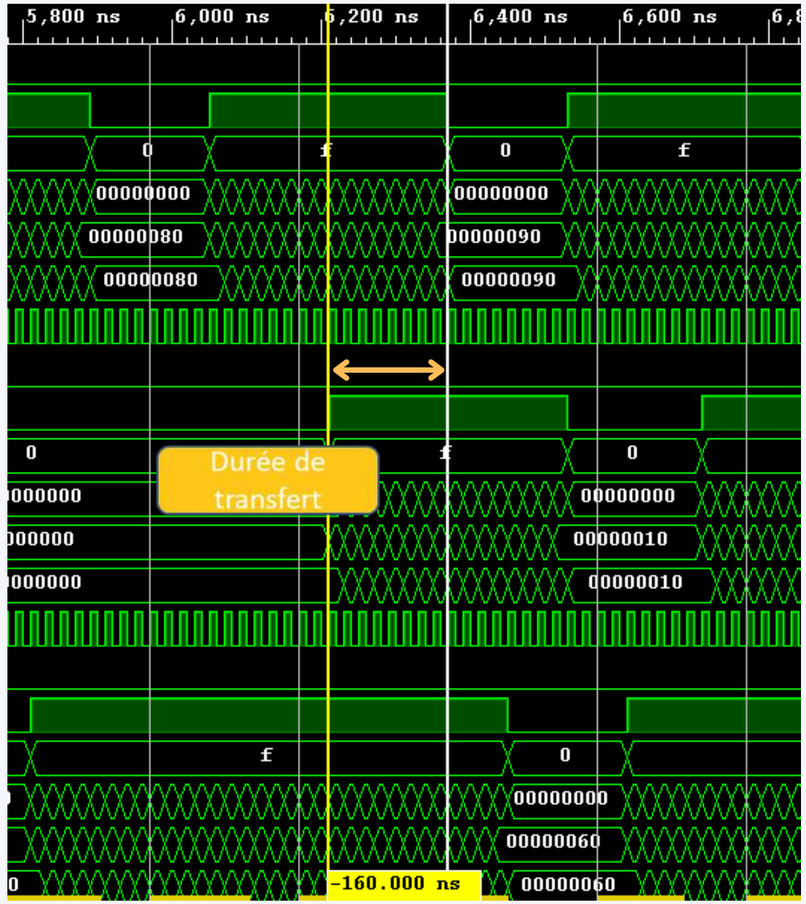


- Le mode shared access est le moins performant .
- Les conflits d’accès pour ce mode font augmenter considérablement les latences.
- les signaux décalés sont plus performant par rapport aux signaux non décalés.

Bande passante :

Ce calcule en divisant le poids des données transférées par la durée du transfert.

$$Bande\_passante = \frac{(Poids\ de\ données\ *3)}{la\ durée\ de\ transfert}$$



Signal 1

Signal d'écriture sur la mémoire 1

Signal 2

Signal d'écriture sur la mémoire 2

Signal 3

Signal d'écriture sur la mémoire 3

Résultat :

Cas : Signaux décalés

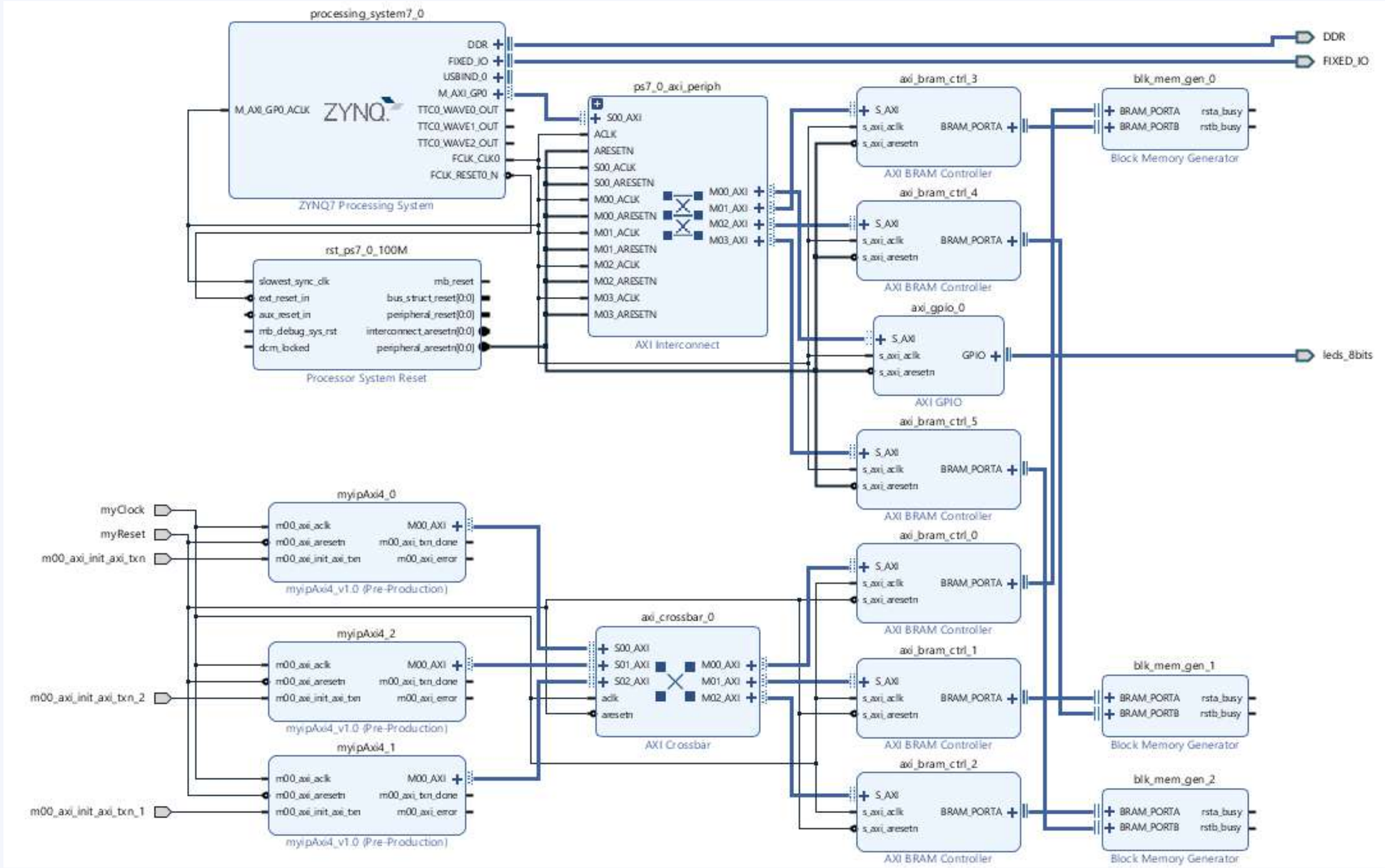
Largeur du data Burst	CrossBar(Mb/s)	Shared Access(Mb/s)
8	50	44
16		47
32		47

Cas : Signaux déclenchés aux même instant

Largeur du data Burst	CrossBar(Mb/s)	Shared Access(Mb/s)
8	50	44
16		47
32		54

Implémentation du NOC 3x3sur Zynq

Implémentation NOC 3x3 sur Zynq



Les informations de placement et routage

Timing :

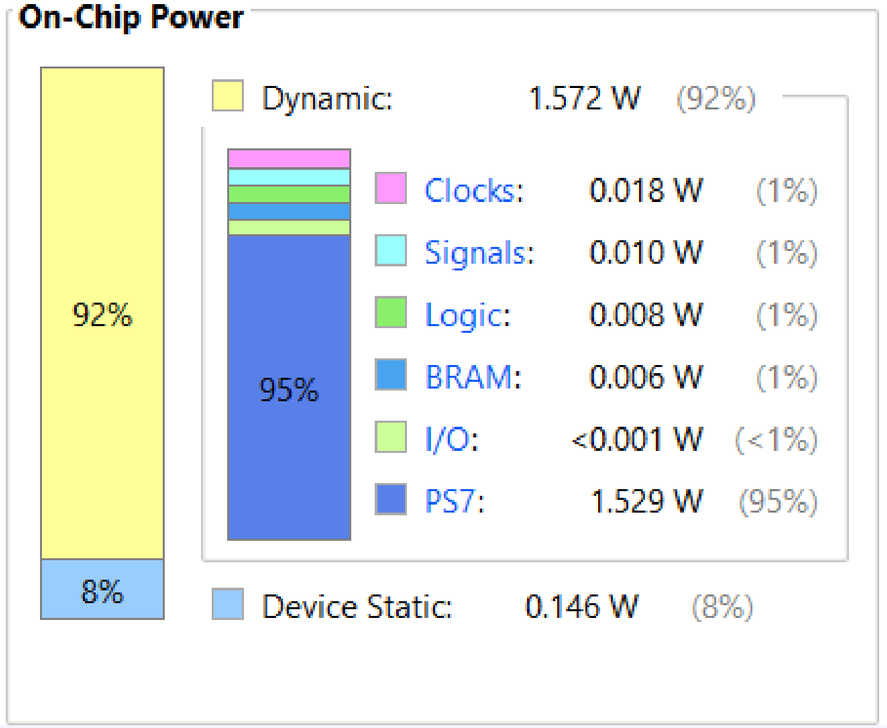
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.300 ns	Worst Hold Slack (WHS):	0.019 ns	Worst Pulse Width Slack (WPWS):	2.520 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	8572	Total Number of Endpoints:	8572	Total Number of Endpoints:	3562

Les informations de placement et routage

Consommation d’énergie :

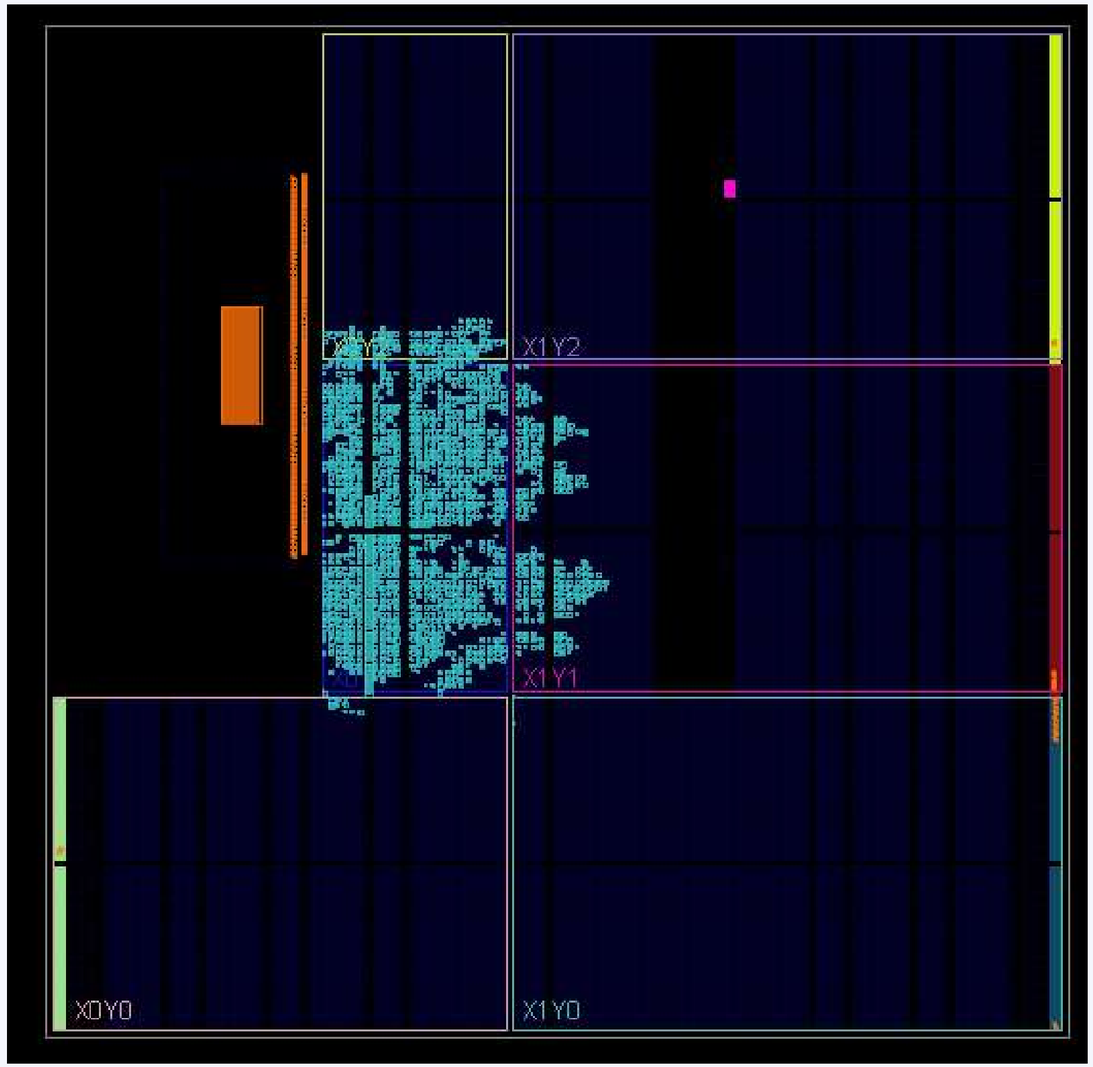
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.718 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**44.8°C**
Thermal Margin:	40.2°C (3.4 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low



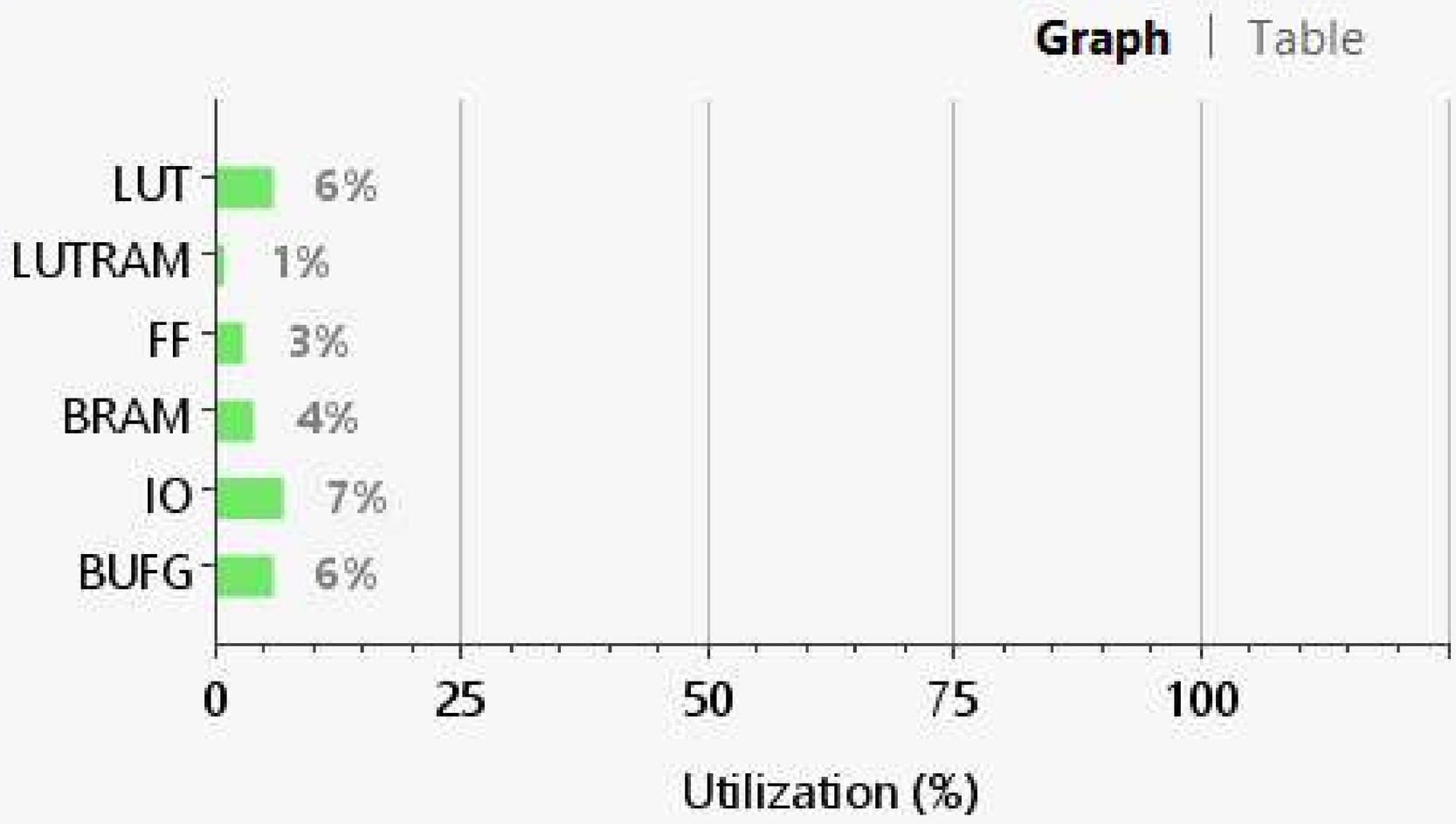
Les informations de placement et routage

Surface occupée :



Les informations de placement et routage

Utilisation matérielle :



Directives de l’optimisation sous Vivado

Performance_ExplorePostRoutePhysOpt

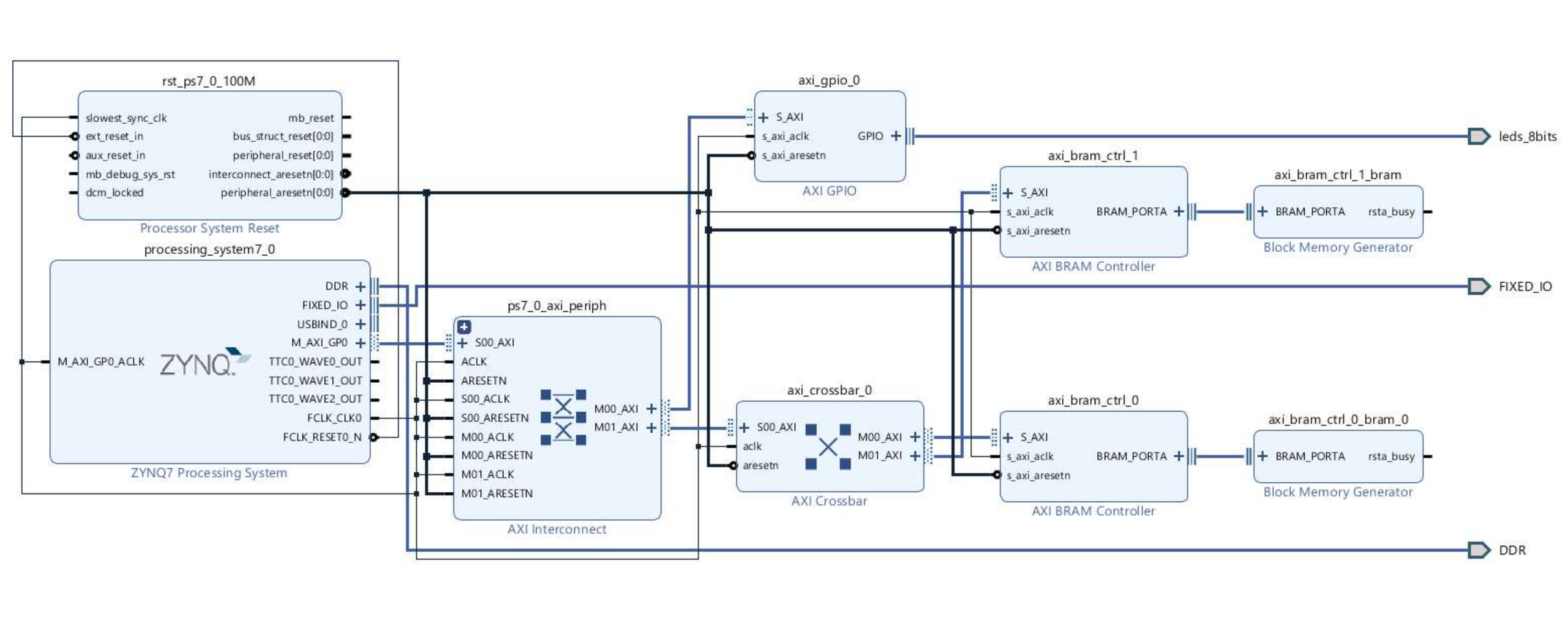
- Explore différentes options de mise en place physique pour optimiser les performances du circuit
- Utilise une combinaison de méthodes de placement et de routage, et des algorithmes de recherche pour trouver la meilleure solution possible

Performance_ExploreWithRemap

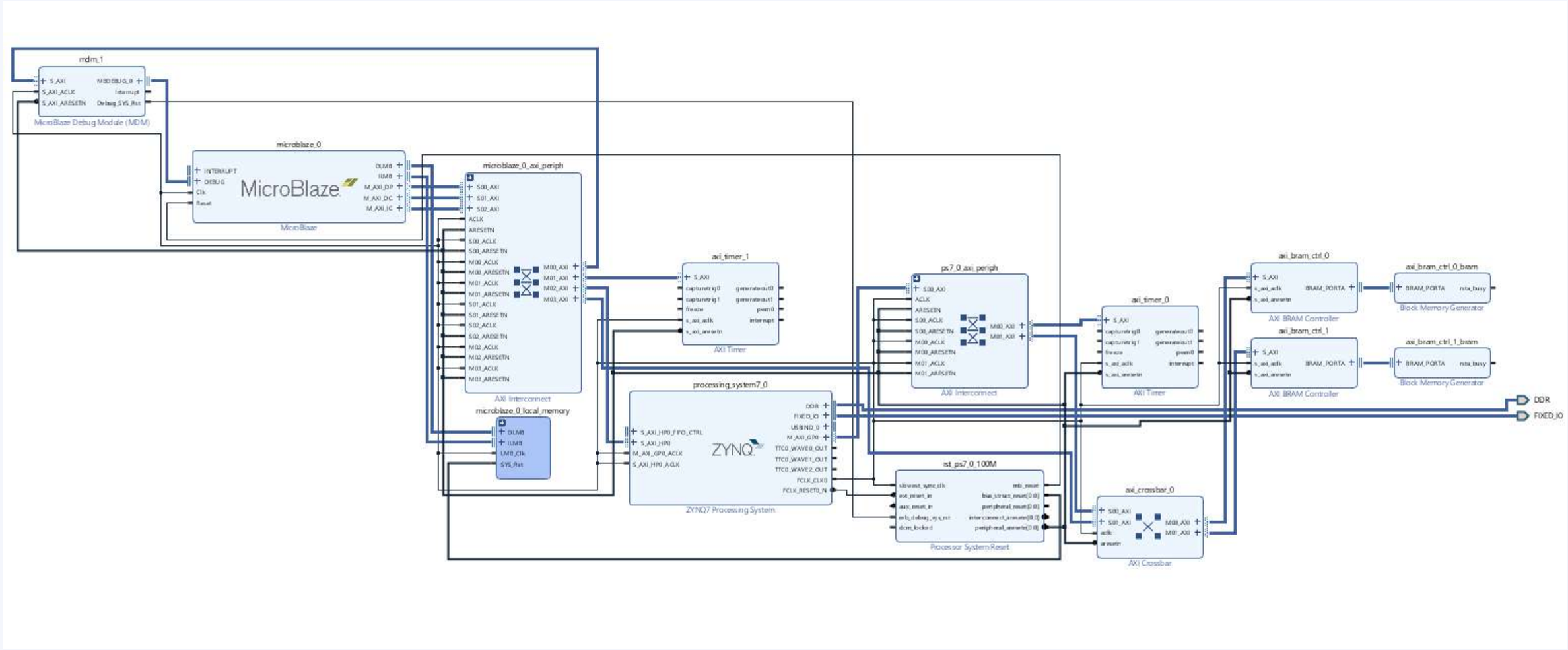
- Le remappage consiste à réorganiser les blocs de la puce pour améliorer les performances en termes de bande passante, de latence et de consommation d'énergie

Multicoeur 3 cœurs : 1 ARM 2 Microblaze

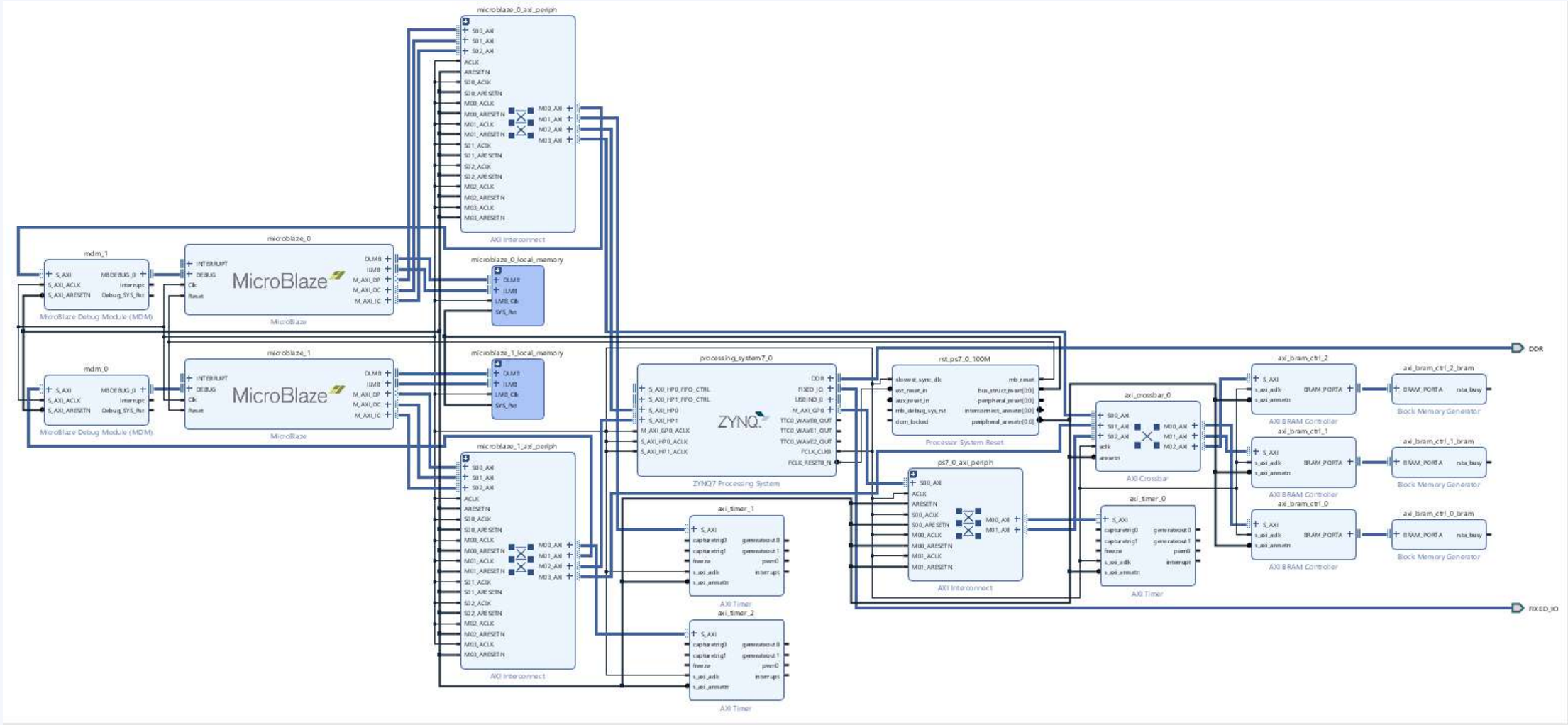
Block design avec Zynq et Crossbar : ARM + Crossbar



Block design NOC 2X2 : 1 Zynq + 1 MicroBlaze

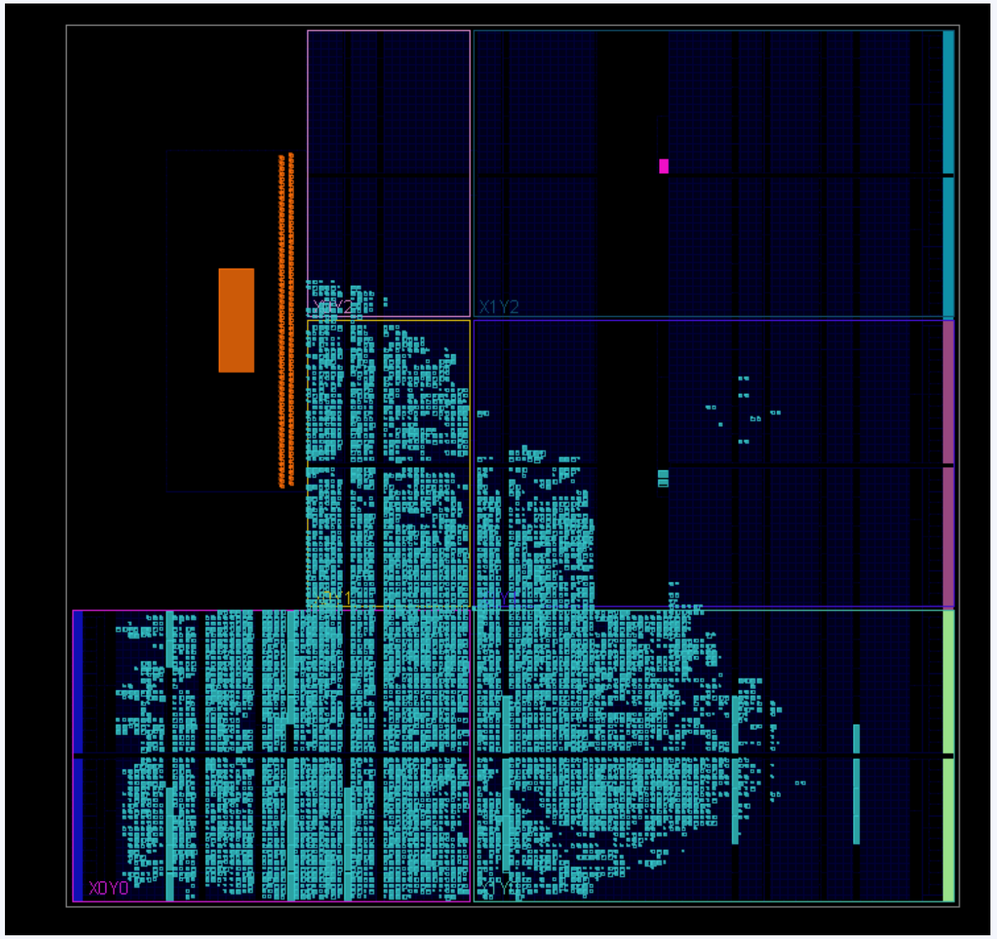


Block design NOC 3X3 : 1 Zynq + 2 MicroBlaze

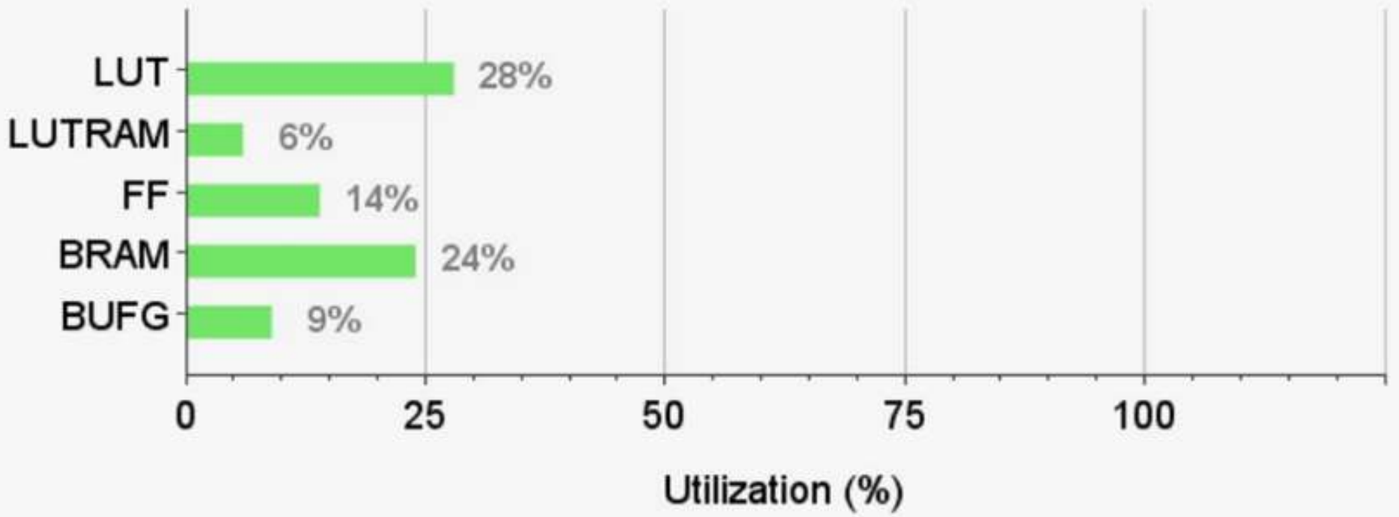


Synthèse des block designs : 1 ARM + 2 MB

Occupation de la surface

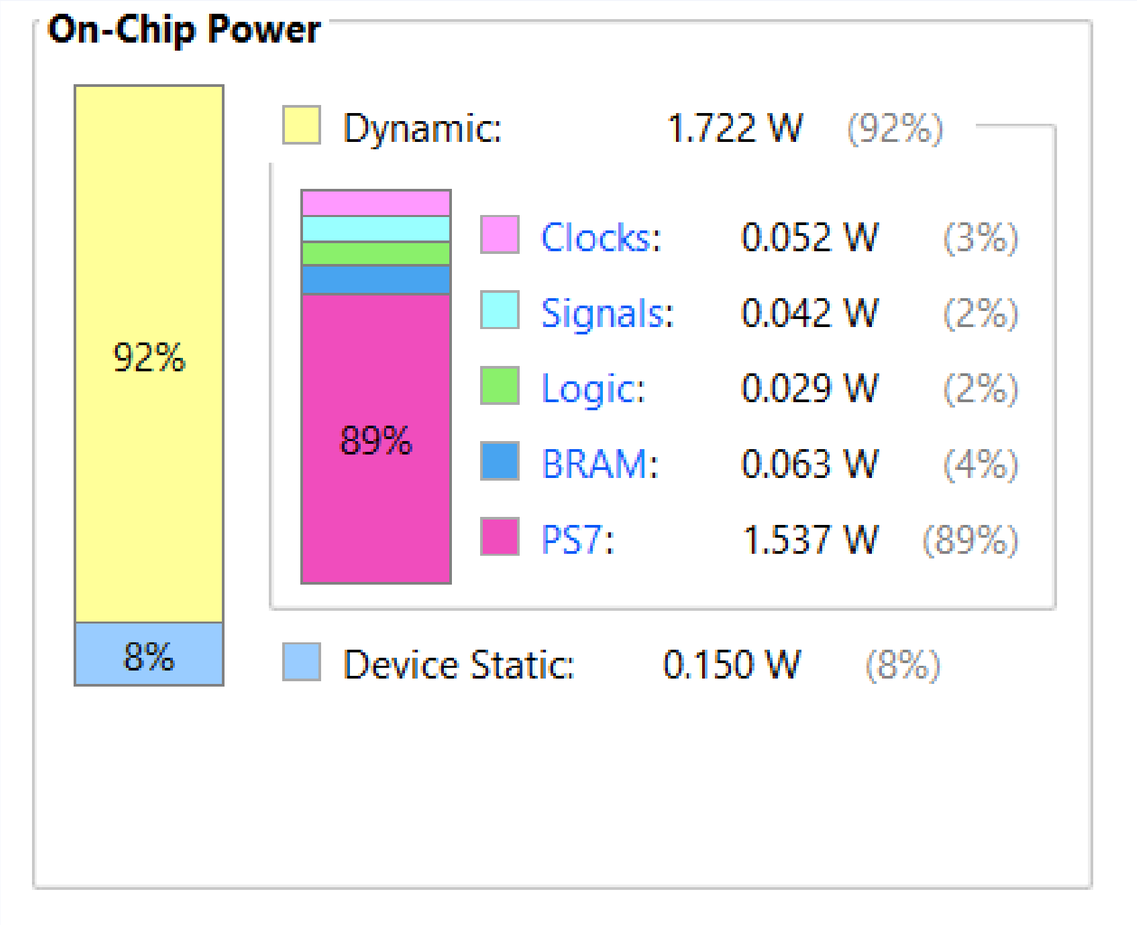


Utilisation matérielle



Synthèse des block designs : 1 ARM + 2 MB

Consommation d'énergie



Synthèse des block designs : 1 ARM + 2 MB

Rapport de timing

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,482 ns	Worst Hold Slack (WHS):	0,012 ns	Worst Pulse Width Slack (WPWS):	3,750 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	43848	Total Number of Endpoints:	43848	Total Number of Endpoints:	16553

All user specified timing constraints are met.

logiciel embarqué

Aspect Robotique et vision

Implémentation d'un planificateur de trajectoire pour le robot Husky : Dijkistra + A*

Implémentation d'un filtre d'estimation d'état

Implémentation d'une fonction de traitement d'image : RGB2GRAY

Aspect IA

Analyse performance et consommation de l'algorithme détection d'objets sur SoC

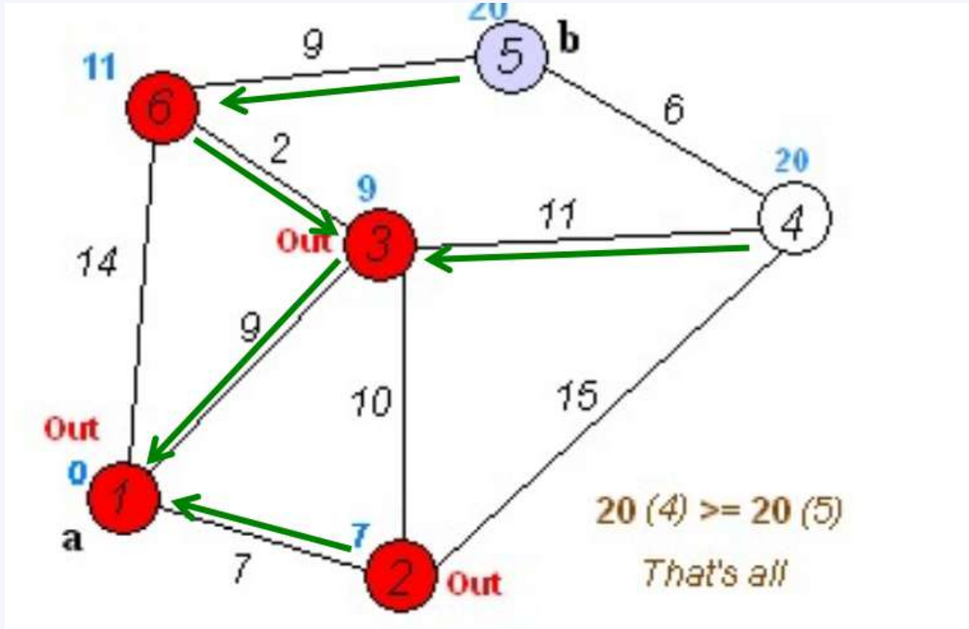
Implémentation d'un classificateur K_means

Planification de trajectoire : Dijkstra

Cet algorithme sert à trouver un chemin optimal entre un point actuel A et un point objectif B dans l'environnement représenté sous la forme d'un graphe en un minimum du temps .

Algorithme 7: Algorithme de Dijkstra

Données : Un graphe orienté pondéré $G = (X, A, W)$ et un sommet $s \in X$
Résultat : Le plus court chemin de s vers tous les autres sommets de G
// V : Tableau stockant les étiquettes des sommets de G
Initialiser V à $+\infty$
$V[s] = 0$
// P : Tableau permettant de retrouver la composition des chemins
Initialiser P à 0
$P[s] = s$
répéter
  // Recherche du sommet x non fixé de plus petite étiquette
  $V_{min} = +\infty$
  pour y *allant de* 1 *à* N **faire**
    si y *non marqué et* $V[y] < V_{min}$ **alors**
      $x \leftarrow y$
      $V_{min} \leftarrow V[y]$
  // Mise à jour des successeurs non fixés de x
  si $V_{min} < +\infty$ **alors**
    Marquer x
    pour *tout successeur* y *de* x **faire**
      si y *non marqué et* $V[x] + W[x, y] < V[y]$ **alors**
        $V[y] = V[x] + W[x, y]$
        $P[y] = x$
jusqu'à $V_{min} = +\infty$



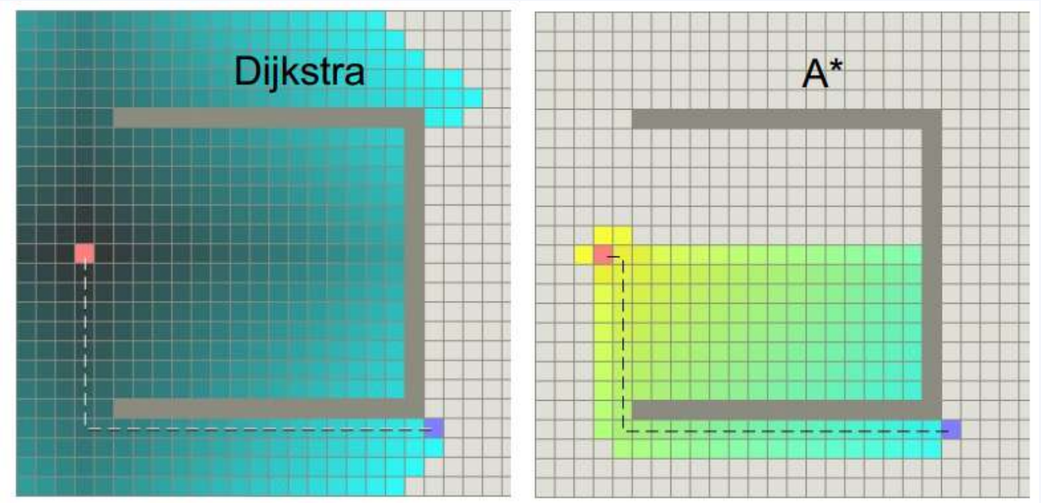
Planification de trajectoire : A*

- Similaire à l'algorithme de Dijkstra
- Algo informé : ajout d'une heuristique estimant la distance d'un point au but (ex : distance euclidienne)
- Traitement en priorité des points ayant la plus faible valeur distance

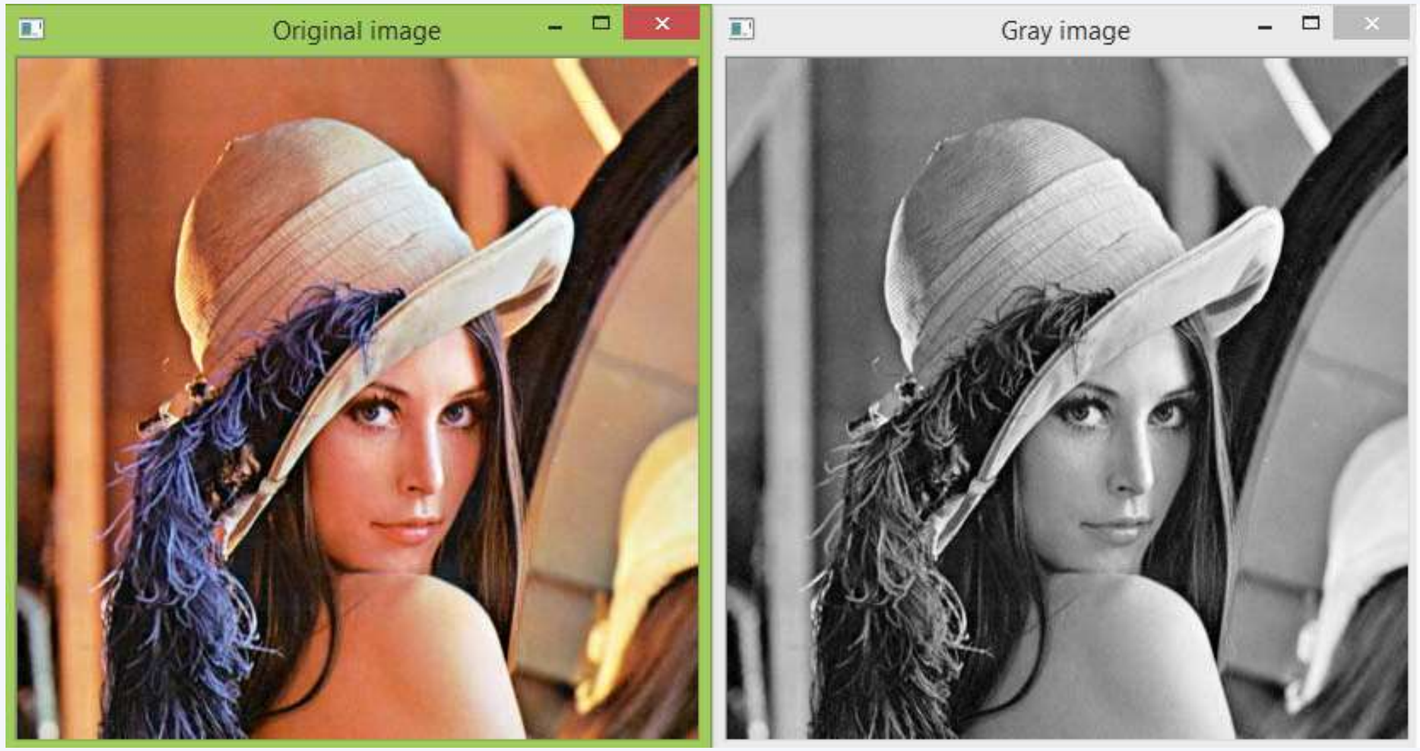
(départ) + μ*heuristique)

⟶ μ règle l'influence de l'heuristique

Permet de ne parcourir qu'une partie des états



Traitement d'image : RGB2GRAY



GRAY = 0,299r + 0,587g + 0,114b

Estimation de la postion

Filtre de Kalman

- Améliorer la précision de la localisation d'un robot en temps réel
- Mettre à jour l'estimation de la position du robot en utilisant les mesures provenant des capteurs
- Naviguer de manière plus précise et éviter les obstacles

Estimation de la postion

Filtre de Kalman

```
// State
double x = 0;
double v = 0;

// Uncertainty
double P = 1;

// Process noise
double Q = 0.1;

// Measurement noise
double R = 0.1;

// Time step
double dt = 1.0;
```

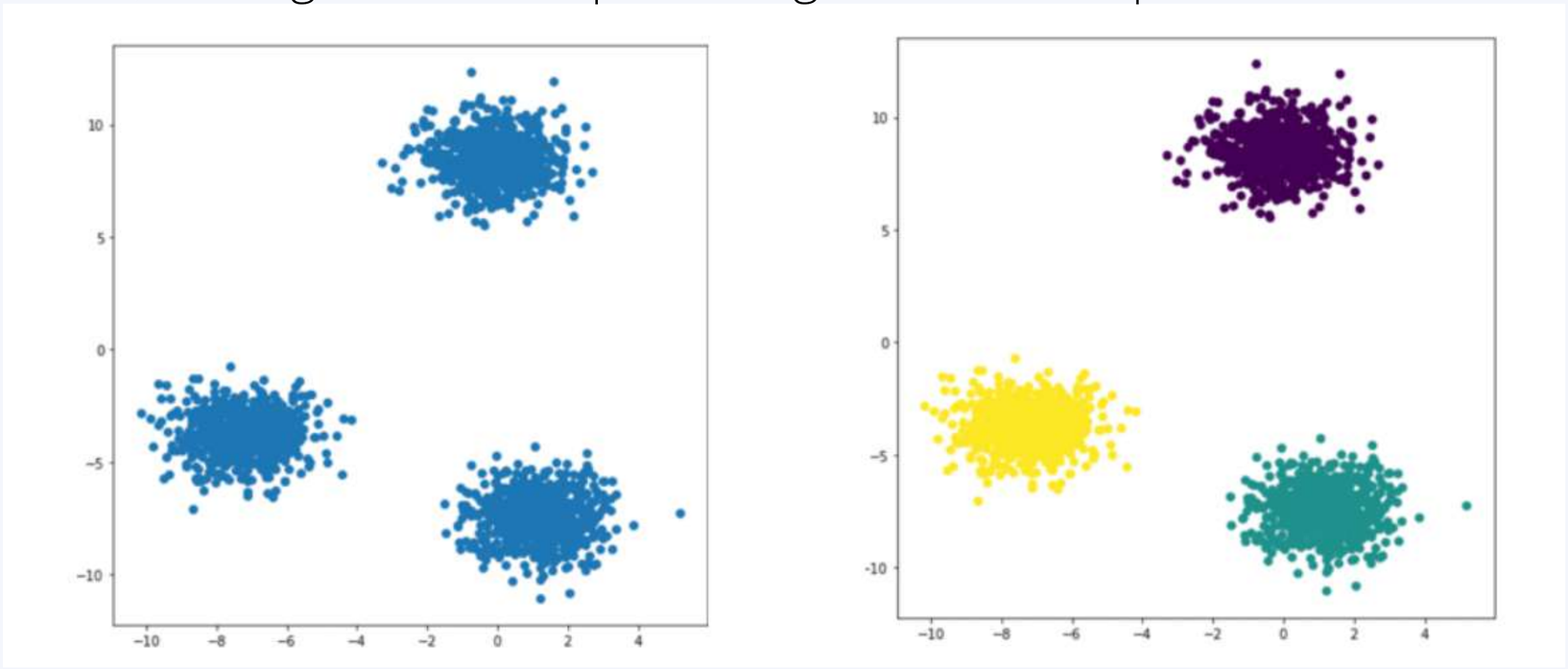
```
void predict() {
    // Prediction
    x += v * dt;
    v += 0;
    P += Q;
}
```

```
void update(double measurement) {
    // Correction
    double residual = measurement - x;
    double S = P + R;
    double K = P / S;
    x += K * residual;
    v += K * residual / dt;
    P *= (1 - K);
}
```

Classification : K-means

Le Principe de l'algorithme des k-means

Étant donnés des points et un entier k, l’algorithme vise à diviser les points en k groupes, appelés clusters, homogènes et compacts. Regardons l’exemple ci-dessous :



Le but

Permettre au robot de reconnaitre des images / des objets / des zones à parcourir

Classification : K-means

```cpp
//KMeans
// les bilbiotheques utilisees
#include <iostream>
#include <vector>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <algorithm>
```

```cpp
class Point
{
private:
    int id_point, id_cluster;
    vector<double> values;
    int total_values;
    string name;
// les fonctions
public:
```

```cpp
class Cluster
{
private:
    int id_cluster;
    vector<double> central_values;
    vector<Point> points;

public:
```

```cpp
class KMeans
{
private:
    int K; // nombre de clusters
    int total_values, total_points, max_iterations;
    vector<Cluster> clusters;
```

Algorithme K-means

Entrée :

K le nombre de cluster à former

DEBUT

Choisir aléatoirement K points. Ces points sont les centres des clusters (nommé centroïd).

REPETER

Affecter chaque point au groupe dont il est le plus proche au son centre

Recalculer le centre de chaque cluster et modifier le centroïde

JUSQU'A CONVERGENCE

OU (stabilisation de l'inertie totale de la population)

FIN ALGORITHME

Classification : K-means

Exemple :

```
4 2 3 10 0
5 6
7 9
1 5.5
5 6
 ** Rupture dans l'iteration 3

** Cluster n° 1
Point 3: 1 5.5
--- Valeurs des clusters: 1 5.5

** Cluster n° 2
Point 1: 5 6
Point 4: 5 6
--- Valeurs des clusters: 5 6

** Cluster n° 3
Point 2: 7 9
--- Valeurs des clusters: 7 9
```

Détection des objets

Objectif :

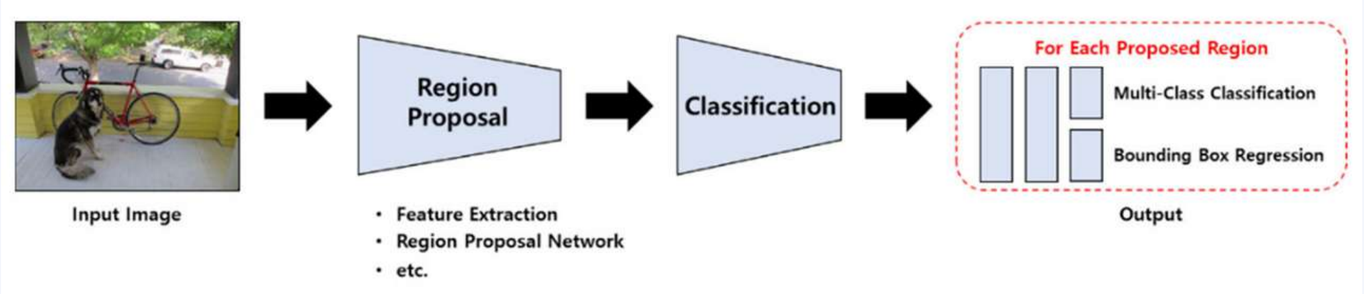
- Algorithme détection des objets choisi -- YOLO
- Analyze de performance et consommation d’ énergie sous différents SoCs:
  Jetson, ARM, FPGA

Application: détection des objets sur robot mobile

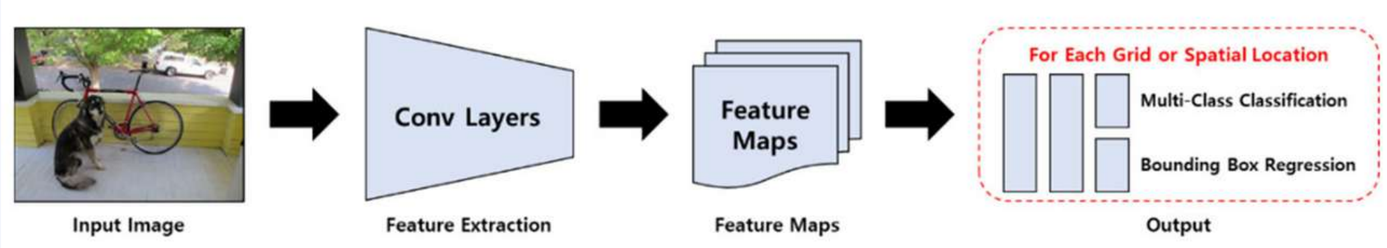
- La détection des objets est l'étude de la classification et de la localisation d'objets cibles.
- Il peut aider les robots mobiles à éviter efficacement les obstacles pour assurer la sécurité de la conduite.

Application: détection des objets sur robot mobile

- Deux stages sont exécutées pendant la détection : la classification et la proposition de région.
- Le détecteur en deux étapes propose d'abord la région d'intérêt puis effectue la classification sur ces régions. -- eg: R-CNN, Fast R-CNN, Faster R-CNN et Mask R-CNN



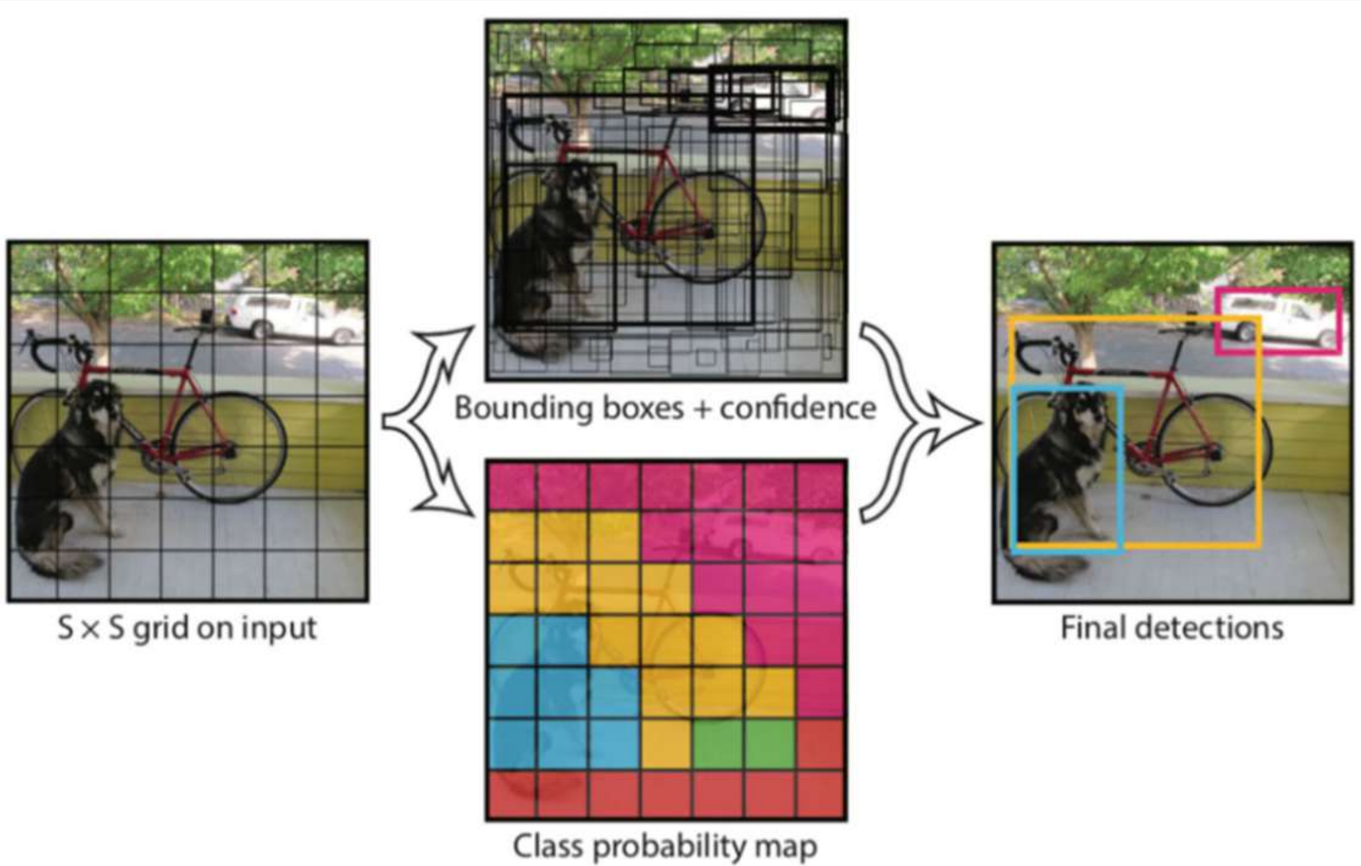
- Le détecteur à une étape effectue la classification et la proposition de région en même temps. -- eg: YOLO, RetinaNet, RefineDet.



Le détecteur à une étape -- YOLO

- YOLO est un détecteur à une étape.

- Grâce à son petit modèle et à sa vitesse de calcul élevée, YOLO est couramment utilisé dans les dispositifs robotiques mobiles embarqués.



Systèmes embarqués Nvidia AI - les Jetsons

- Les plates-formes Jetson sont des systèmes embarqués qui exécutent des algorithmes d'IA en parallèle sur des GPUs.

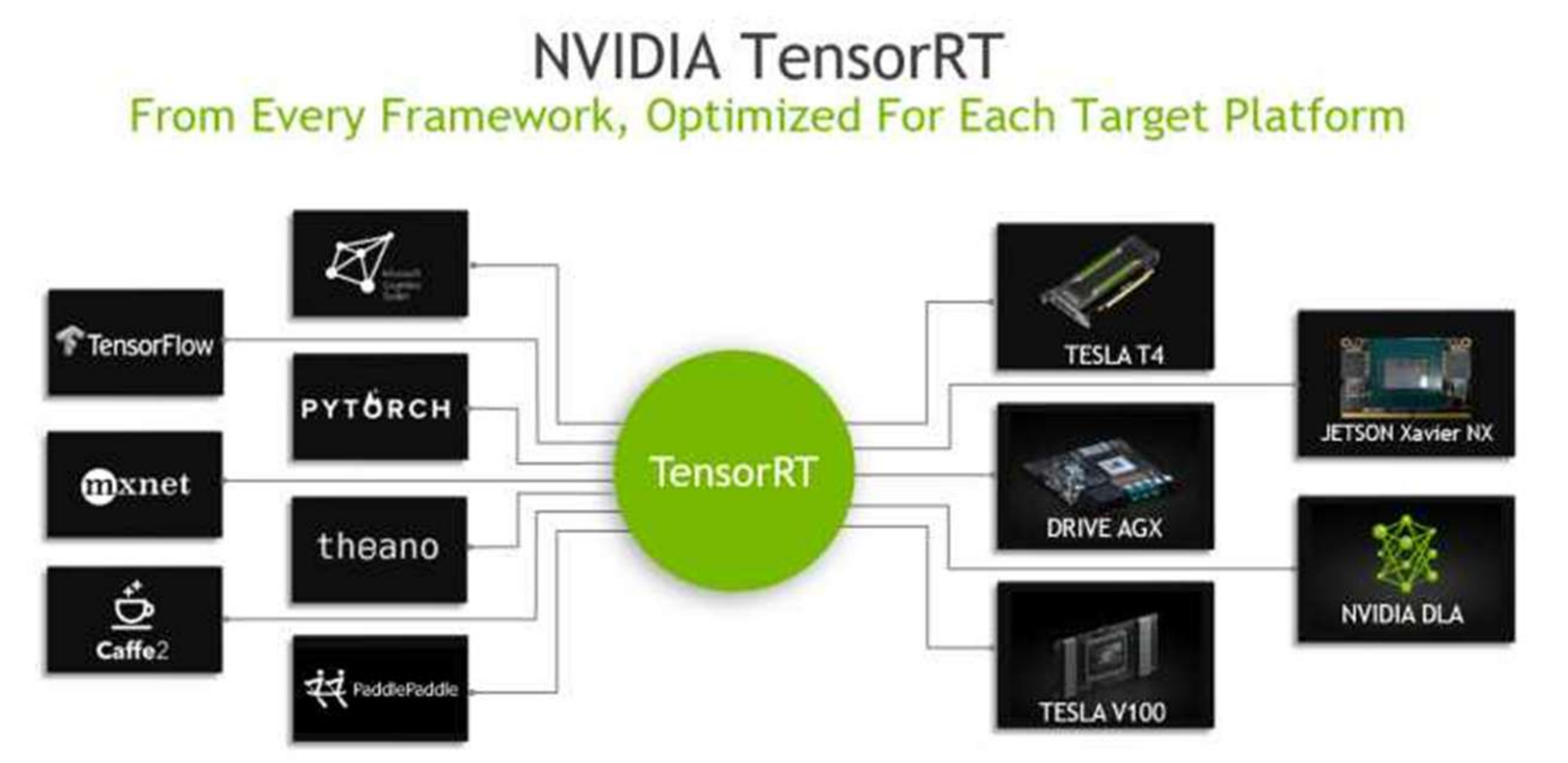
Jetson Nano

Jetson TX2

Jetson AGX Xavier

Systèmes embarqués Nvidia AI - les Jetsons

- Par rapport aux FPGAs, le Jetson s'appuie sur son système d'opération Ubuntu, qui permettre d'appliquer rapidement des algorithmes d'IA dans des cadres de calcul courants (Tensorflow, Pytorch, etc.).



Systèmes embarqués Nvidia AI - les Jetsons

- Les performances des différents modèles de Jetson.

	Jetson NANO	Jetson TX2			Jetson AGX Xavier	
		4 GB	8 GB	Industrial	8 GB	16 GB
GPU	128-Core Maxwell GPU with CUDA Cores	256-Core Pascal GPU with CUDA Cores			384 Core Volta + NVDLA	512 Core Volta + NVDLA
CPU	Quad-core ARM Cortex-A57	Quad-core Arm Cortex-A57 Quad-core ARM A57 complex			6-core Carmel ARM CPU 1.3 GHZ	8-core Carmel ARM CPU 2.26 GHz
Memory	4 GB 64-bit LPDDR4	4 GB 128-bit LPDDR4	8 GB 128-bit LPDDR4		8 GB 256-bit LPDDR4x	16 GB 256-bit LPDDR4x

Analyse des performances et consommation énergie de l'algorithme YOLO sur Jetson

- L'analyse comparative de YOLOs sur Jetson AGX Xavier 16 GB effectuée par chercheurs *.
- Les indications de performance sont mesurées pour YOLOv4-Tiny sous le COCO dataset, sous different cadre d'éxécution:

Framework (YOLOv4-Tiny in the COCO Dataset)	Precision	Average CPU Utilization (%)	Average GPU Utilization (%)	Average Accuracy (%)	Average Latency (ms)	Average Power (W)
TensorFlow	Float32	27.99	52.19	Person: 33.33 Car: 60.73	28.38	14.38
TF-Lite	Float32	20.32	0.13	Person: 33.96 Car: 60.38	600.67	9.71
TF-TRT		28.13	51.08	Person: 33.95 Car: 60.46	30.25	11.01
TRT		31.18	39.76	Person: 34.70 Car: 60.31	18.02	12.06
TF-Lite	Float16	20.57	0.13	Person: 33.76 Car: 60.10	604.10	9.48
TF-TRT		31.81	38.62	Person: 33.76 Car: 60.93	24.19	10.10
TRT		31.07	42.20	Person: 34.02 Car: 61.01	17.85	12.40

*: Dong-Jin Shin and Jeong-Joon Kim. A deep learning framework performance evaluation to use yolo in nvidia jetson platform. Applied Sciences, 12(8):3734, 2022.

Analyse des performances et consommation énergie de l'algorithme YOLO sur Jetson

- Selon les données, YOLOv4-Tiny fonctionne sur Jetson AGX Xavier 16 GB avec une consommation d'énergie d'environ 10W et une latence d'environ 30ms, varié pour différents cadre de calcul.

Framework (YOLOv4-Tiny in the COCO Dataset)	Precision	Average CPU Utilization (%)	Average GPU Utilization (%)	Average Accuracy (%)	Average Latency (ms)	Average Power (W)
TensorFlow	Float32	27.99	52.19	Person: 33.33 Car: 60.73	28.38	14.38
TF-Lite	Float32	20.32	0.13	Person: 33.96 Car: 60.38	600.67	9.71
TF-TRT		28.13	51.08	Person: 33.95 Car: 60.46	30.25	11.01
TRT		31.18	39.76	Person: 34.70 Car: 60.31	18.02	12.06
TF-Lite	Float16	20.57	0.13	Person: 33.76 Car: 60.10	604.10	9.48
TF-TRT		31.81	38.62	Person: 33.76 Car: 60.93	24.19	10.10
TRT		31.07	42.20	Person: 34.02 Car: 61.01	17.85	12.40

*: Dong-Jin Shin and Jeong-Joon Kim. A deep learning framework performance evaluation to use yolo in nvidia jetson platform. Applied Sciences, 12(8):3734, 2022.

Analyse des performances et consommation énergie de l'algorithme YOLO sur Jetson

- Les performances et consommation énergie de l'algorithme YOLOv3 et YOLOv3-tiny sont aussi compare sous différent type de appareil Jetson*.

	Models	Accelerator -Based SBCs	Mean Confidence (%)	FPS	CPU Usage (%)	Memory Usage (GB)	Power (W)	Time (s)
Video1	YOLOv3	RPi + NCS2	99.3	2.5	4.3	0.33	6.0	690
		Nano	99.7	1.7	26.5	1.21	7.9	967
		NX	99.7	6.1	22.5	1.51	15.2	256
	YOLOv3-tiny	RPi + NCS2	0	18.8	15.5	0.11	6.5	121
		Nano	59.7	6.8	28.8	1.00	7.2	236
		NX	59.7	41.1	30.5	1.33	13.5	46
Video2	YOLOv3	RPi + NCS2	85.8	2.5	9.8	0.41	6.2	496
		Nano	71.5	1.6	28.8	1.36	8.0	587
		NX	71.5	5.9	26.8	1.69	15.2	162
	YOLOv3-tiny	RPi + NCS2	61.5	19.0	24.8	0.18	6.8	162
		Nano	54.1	6.6	37.3	1.16	7.4	152
		NX	54.1	35.6	55.5	1.47	13.2	31

*: Haogang Feng, Gaoze Mu, Shida Zhong, Peichang Zhang, and Tao Yuan. Benchmark analysis of yolo performance on edge intelligence devices. Cryptography, 6(2):16, 2022.

Analyse des performances et consommation énergie de l'algorithme YOLO sur Jetson

- Selon les données expérimentaux *, la consommation d'énergie varie de 10W à 15W avec la vitesse d'exécution 30-40 FPS.

	Models	Accelerator -Based SBCs	Mean Confidence (%)	FPS	CPU Usage (%)	Memory Usage (GB)	Power (W)	Time (s)
Video1	YOLOv3	RPi + NCS2	99.3	2.5	4.3	0.33	6.0	690
		Nano	99.7	1.7	26.5	1.21	7.9	967
		NX	99.7	6.1	22.5	1.51	15.2	256
	YOLOv3-tiny	RPi + NCS2	0	18.8	15.5	0.11	6.5	121
		Nano	59.7	6.8	28.8	1.00	7.2	236
		NX	59.7	41.1	30.5	1.33	13.5	46
Video2	YOLOv3	RPi + NCS2	85.8	2.5	9.8	0.41	6.2	496
		Nano	71.5	1.6	28.8	1.36	8.0	587
		NX	71.5	5.9	26.8	1.69	15.2	162
	YOLOv3-tiny	RPi + NCS2	61.5	19.0	24.8	0.18	6.8	162
		Nano	54.1	6.6	37.3	1.16	7.4	152
		NX	54.1	35.6	55.5	1.47	13.2	31

*: Haogang Feng, Gaoze Mu, Shida Zhong, Peichang Zhang, and Tao Yuan. Benchmark analysis of yolo performance on edge intelligence devices. Cryptography, 6(2):16, 2022.

Comparaison FPGA , CPU et GPU

- Analyse de la performance de YOLOv3-tiny en comparant CPU, GPU et carte FPGA (ZYNQ - 7035) est fait dans l'article *.

Table 1: comparison to GPU and CPU based AI accelerator

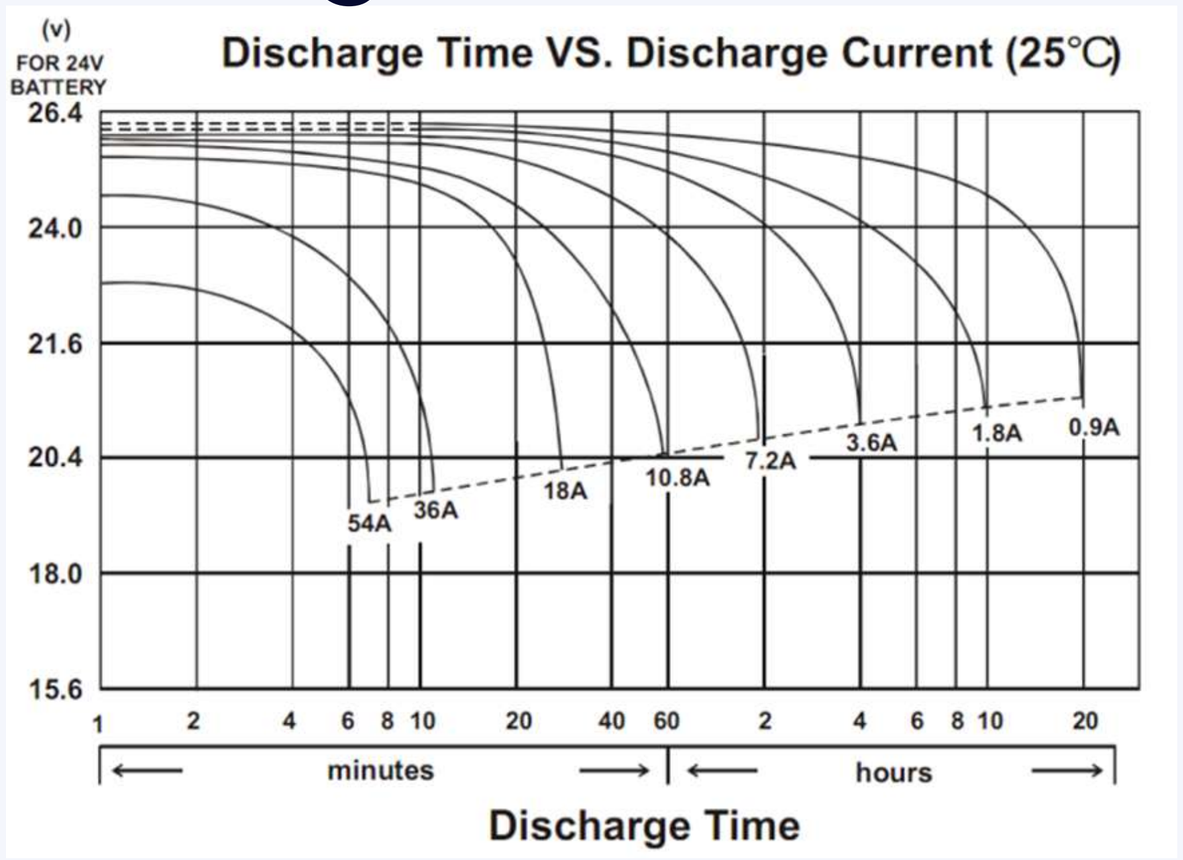
	[12]	[22]	This Work
Platform	Ryzen 5 3600	Jetson Nano	ZYNQ-7035
Platform Type	CPU	GPU	ARM + FPGA
Clock Freq	3.6 GHz	1300MHz	100MHz
Latency	7937ms	40ms	192ms
Power	-	10W	3.71W

- On constate que le FPGA (ZYNQ - 7035) ont une consommation d'énergie plus faible (3.7W) que le GPU (Jetson Nano) (10W).
- Mais le FPGA présente une latence plus élevée de 192 ms, soit près de cinq fois plus que le GPU.
- La latence élevée a un impact significatif sur la réactivité de la détection et doit être choisie avec soin en fonction du scénario réel.

*: Qilin Xiong, Chun Liao, Zhenhong Yang, and Wanlin Gao. A method for accelerating yolo by hybrid computing based on arm and fpga. In 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence, pages 1–7, 2021.

Analyse de la consommation d'énergie de l'algorithme de détection des objets sur le robot Husky

- Supposons que le YOLOv4-tiny est la seule application fonctionnant sur le robot.
- La puissance moyenne nécessaire est ~15W.
- Le courant de décharge en 25'C est
  I = P/U = 15W/24V = 0.6 A.
- La capacité de la batterie(>24V) est d'environ 0.9A*10h = 9 A.h.
- -- Le temps de fonctionnement d'un seule application YOLOv4-tiny sur le robot Husky est 9A.h / 0.6 A = 15h.



La courbe de décharge de la batterie du robot Husky.

Discussion sur l’application détection des objets

- L'application des YOLO sur une plateforme FPGA nécessite une puissance moindre. Mais une durée de fonctionnement de 15h est acceptable avec Jetson, comme le robot peut prendre l'energie s'il est en repos.

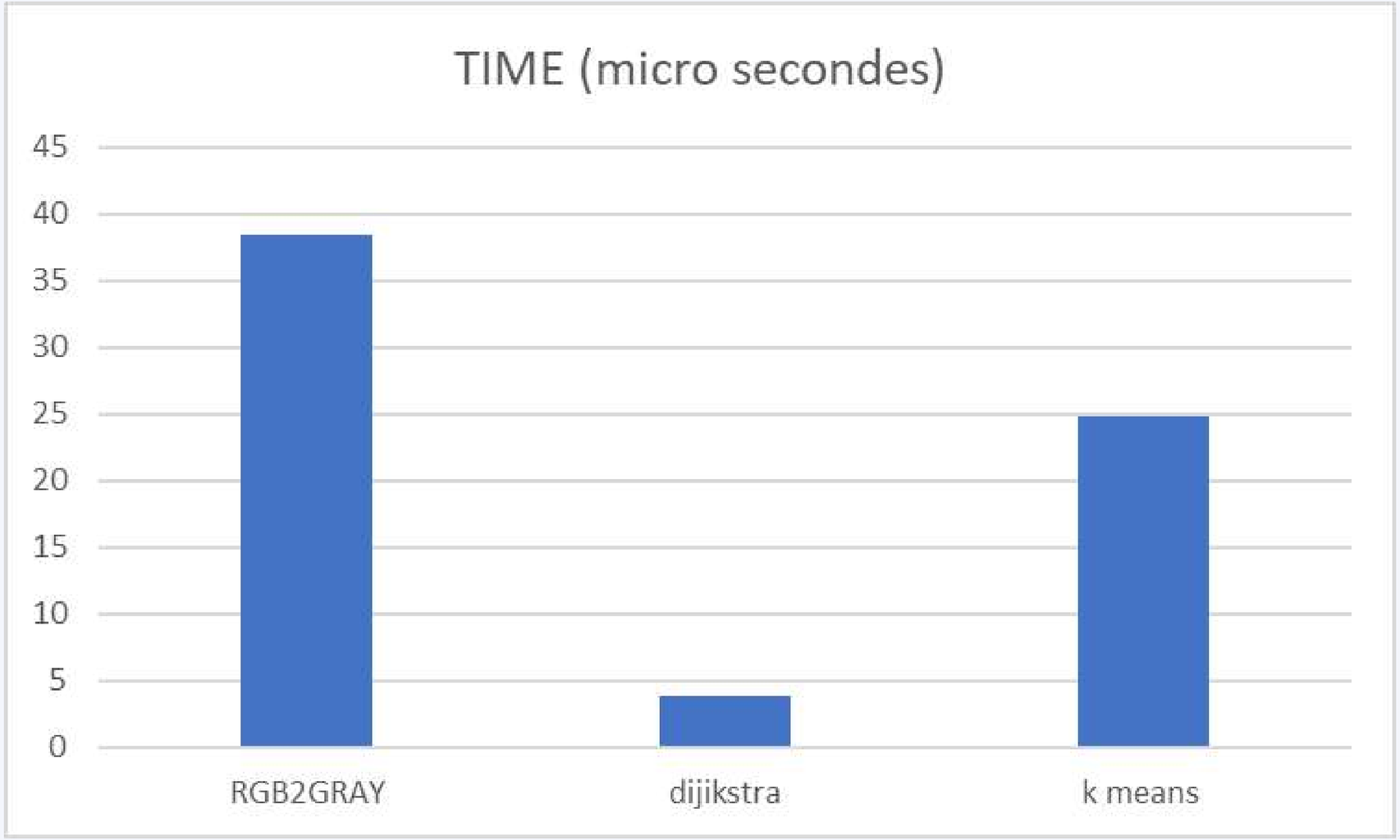
- Cependant, la fréquence est plus faible sur le FPGA, et il nécessite aussi plus de technique dans la programmation des packages IPs.

- Avec la plateforme Jetson, son système d'opération Ubuntu permet une transformation rapide des algorithmes de détection d'objets.

- Le FPGA sont donc plus adaptés au déploiement d'algorithmes déjà établis dans des produits commerciaux, tandis que les Jetsons conviennent mieux aux tests expérimentaux.

Aspect énergitique

Temps de calcul sur ARM



Aspect énergétique de l'algorithme Dijkstra

Outils utilisés :

- **SimpleScalar** est un simulateur de microprocesseur superscalaire qui permet l’évaluation de performances de différentes architectures.
- **Instruction Set Simulator – ISS** un modèle de simulation qui imite le comportement d'un microprocesseur en « lisant » des instructions et en maintenant des variables internes qui représentent les registres du processeur

Processeur :

- **Cortex A7** c'est un processeur d'architecture ARM très basse consommation.

1/ Génération de jeu d'instruction à partir du code C :

Mesurer certains paramètres de performance de programme en utilisant un simulateur de processeur (Instruction Set Simulator – ISS)

⟶ L'algorithme Dijktra est compilé dans un jeu d'instructions interprétable par le simulateur . Le programme est compiler avec sslittle-na-sstrix-gcc :

```
sslittle-na-sstrix-gcc dijkstra.c –o dijkstra.ss
```

2/ Profiling:

Identifier pour un programme donné la classe (le type) des instructions sollicitées et le pourcentage d'instructions exécutées issues de cette classe.
La commandel **sim-profile** permet d'analyser l'exécution d'un programme et de générer les pourcentages d'utilisation de chaque classe d'instructions.

```
sim-profile -iclass dijkstra.ss input.dat
```

Résultat de profiling :

Classe d'instructions	Nombre d'instructions	Pourcentage d'instruction (%)
Load	14786552	26.94
Store	4345374	7.92
Unconditional branch	437907	0.80
Conditional branch	9381122	17.09
Int computation	25930417	47.25
Fp computation	0	0.00

les catégorie d'instructions qui nécessitent une amélioration de performances :

Int computation - Load - Conditional branch

3/ Mesure de l' IPC en fonction de la taille du cache L1, pour dijkstra :

- On va faire une Simulation avec sim-outorder pour l'application dijkistra.

- A chaque fois, on va simuler en variant simultanément la taille des caches L1 d'instructions et de données de 1KB à 16KB tout en gardant la taille de L2 constante et égale à 512KB, et en précisant les caractéristiques du coeur Cortex A7. Pour finalement, déterminer la configuration de L1 qui donne les meilleures performances.

```
sim-outorder -bpred :bimod 256 -fetch :ifqsize 4 -decode :width 2 -issue :width 4
-issue :inorder false -commit :width 2 -ruu :size 2 -lsq :size 8 -res :ialu 1 -res :imult 1 -
res :fpalu 1 -res :fpmult 1 -res :memport 2 -cache :il1 il1 :16 :32 :2 :l -cache :dl1 dl1 :16 :32 :2 :l
-cache :il2 dl2 -cache :dl2 ul2 :2048 :32 :8 :l
```

3/ Mesure de l' IPC pour dijkstra :

IPC c'est le nombre des instructions par cycle (IPC) .
La variation de cache L1 implique le changement de ce paramètre

Taille cache L1	IPC
1	1,6186
2	1,8425
4	1,9047
8	1,9312
16	1,955

4/ Efficacité énergétique :

1. La puissance consommée :

- Le calcule de la puissance consommée par le processeur à la fréquence maximale est donné par la formule :

$$\textbf{Consommation énergie}_{max}(mW) = \textbf{consommations énergétiques}\left(\frac{mW}{MHz}\right) \times F(GHz).10^3$$

- Les consommations énergétiques du Cortex A7 est de 0.10 mW/MHz .
- En technologie 28 nm, la fréquence maximale du Cortex A7 est de 1.0 GHz.

⟶ Consommationénergie_maxA7(mW) = 100mW

2. L'efficacité énergétique :

- Le calcule de l'efficacité énergétique de chaque processeur à la fréquence maximale est donné par la formule :

$$EfficacitéEnergétique_{max}\left(\frac{1}{mW}\right) = \frac{ICP}{Consommationénergie_{max}(mW)}$$

4/ Efficacité énergétique :

- En utilisant les valeurs ICP de chaque configuration de L1 et en appliquant la formule, les valeurs de l'efficacité énergétique maximale pou l'application DIJKSTRA sont :

Taille cache L1	EÉ max (1/mW)
1	0.016186
2	0.018425
4	0.019047
8	0.019312
16	0.01955

Merci de votre attention