CONSOMMATION D'ENERGIE DES SOCS RAPPORT TP 1

1 INTRODUCTION

Le but de ce TP est de mesurer les consommations statique et dynamique d'un même circuit avec différentes architectures. On en déduira un modèle mathématique permettant de prévoir la consommation d'un circuit. Les cartes utilisées sont des FPGA de la famille Cyclone III.

2 MODELE DE CONSOMMATION

2.1.1 MODELE DE CONSOMMATION STATIQUE

Pour déterminer un modèle de consommation statique, on étudie la consommation d'un bloc additionneur dont on fait varier le nombre de bits, de 32 à 128.

On remarque que la puissance consommée de l'additionneur augmente de façon linéaire par rapport au nombre de bits.

En comparant la puissance statique des entrées et celle de l'opérateur, on remarque que l'opérateur consomme systématiquement plus.

	Puissance statique		
NB BITS	Pcore	Pio	Ptotal
32.00	46.13	17.87	64
64.00	51.8	30.23	82.03
128.00	88.91	51.46	140.37

Figure 1- Tableau puissance statique additionneur (mW)

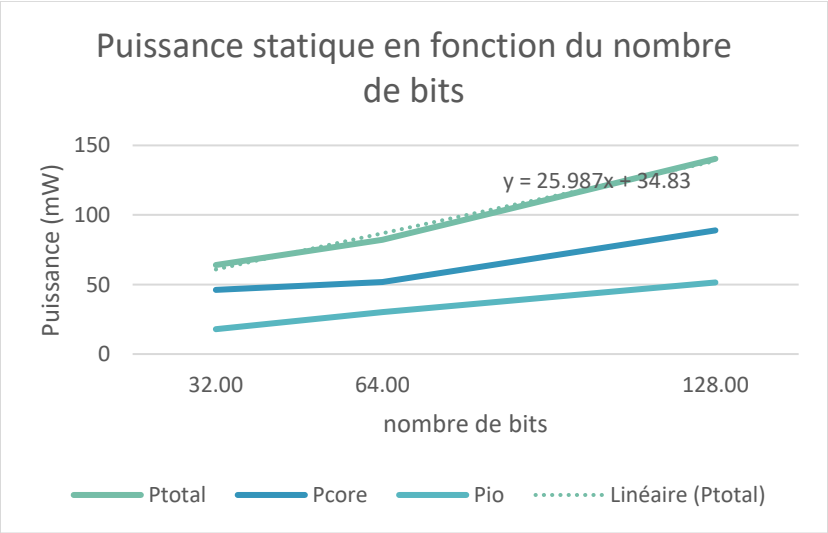


Figure 2- Graph puissance statique en fonction du nombre de bit additionneur

2.1.2 MODELE DE CONSOMMATION DYNAMIQUE

Pour déterminer un modèle de consommation dynamique, on reprend le même bloc additionneur que précédemment. On ajoute au test une séquence de valeurs aléatoires pour chaque entrée, que l'on génère dans un fichier .saf (*signal activity* file).

On remarque que comparé au statique, les IO consomment beaucoup plus que l'additionneur. Cela est due au pics de consommation des IO lors des commutations.

La consommation statique est constante en fonction de la fréquence, ce qui est cohérent.

La consommation dynamique suit une tendance linéaire en fonction de la fréquence, ce qui corrobore la formule théorique $P = c * V^2 * f + P_s$, avec ici V constant.

	32 BITS Statique	32 BITS Dynamique		
Frequence (Mhz)	Pcore_statique	Pcore_dynamique	Pio_dynamique	Ptotal
25.00	46.16	1.59	44.96	92.71
50.00	46.19	3.27	72.83	122.29
100.00	46.25	6.41	126	178.66
200.00	46.39	13.02	236.25	295.66
300.00	46.53	19.53	355.46	421.52

Figure 3- Tableau consommation dynamique additionneur (mW)

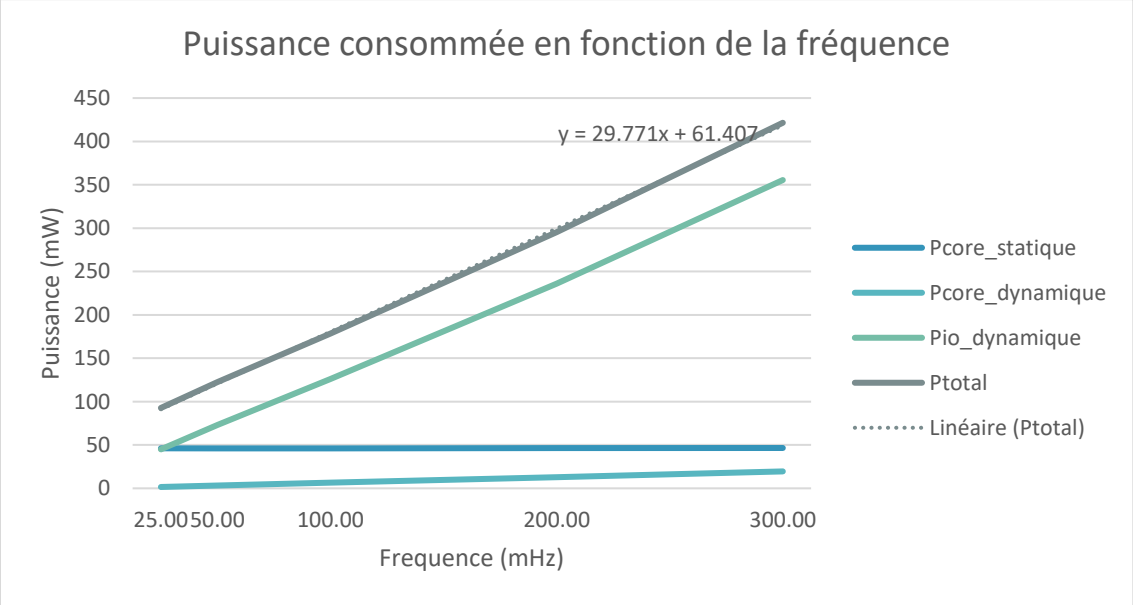


Figure 4- Graph consommation dynamique en fonction de la fréquence additionneur

3 PARALLELISME ET PIPELINE

Dans cette partie, on s'intéresse à la consommation en fonction de l'architecture d'implémentation. En particulier, comment la consommation évolue en ajoutant du pipelining ou en utilisant des circuits dédiés.

3.1.1 CONSOMMATION EN PIPELINING

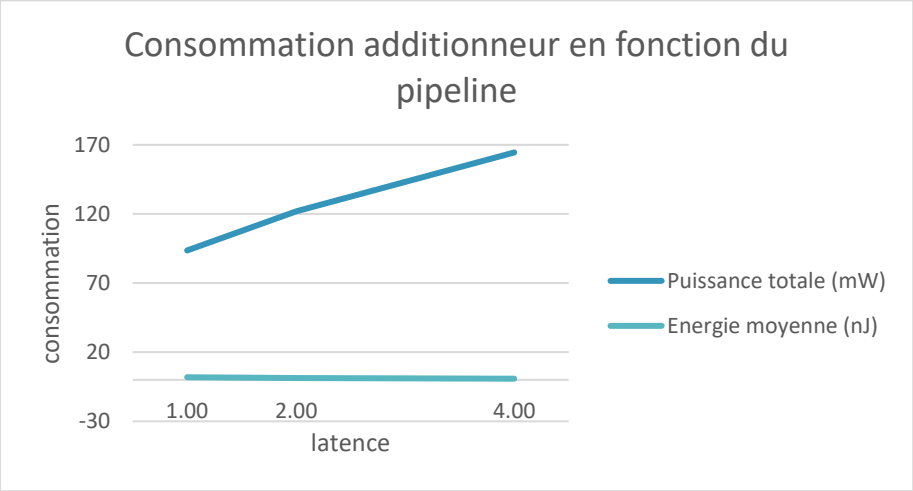
On cherche à évaluer l'impact du pipelining sur la consommation d'un additionneur 32 bits.

On observe les données sur 20ns pour 1 cycle, 10ns pour 2 cycles, et 5ns pour 4 cycles.

On remarque que l'énergie moyenne diminue quand on augmente la latence, mais que la puissance instantanée augmente.

Cela s'explique par le temps entre chaque cycles qui diminue plus vite que n'augmente la puissance ($E = P * t$).

Pipeline (nb cycles	fréquence (MHz)	Puissance totale (mW)	Energie moyenne (nJ)
1.00	50	93.61	1.8722
2.00	100	122.04	1.2204
4.00	200	164.51	0.82255



3.1.2 COMPARAISON : LUT OU DSP

Dans cette partie on étudie la consommation d'un multiplieur. On peut implémenter ce circuit de deux façons. D'abord en utilisant les blocs logiques de la FPGA (LUT), ou en utilisant un circuit dédié DSP implémenté dans le hardware de la carte. Ce dernier devrait être plus rapide et moins consommant puisqu'il s'agit d'un circuit dédié à ce genre d'opérations, il ne nécessite donc pas de beaucoup de routage entre ls blocs logiques.

Multiplicateur sur LUT			Multiplicateur sur DSP		
Pconso (mW)	Treponse (ns)	Econso (nJ)	Pconso	Treponse	Econso
1105.91	31.46	35	456.03	19.298	9

Le circuit DSP est moins consommant et plus rapide que l'implémentation avec les LUTs. Ce résultat est pertinent puisque le circuit DSP est dédié à ce genre d'opérations. Utiliser les LUTs rajoute du routage entre les différents blocs, ce routage est source de latence et d'une plus grosse consommation d'énergie.

4 ETUDE DE CAS : FILTRE DE TRAITEMENT D'IMAGE

On cherche à implémenter un filtre simple de traitement d'image et à évaluer sa consommation.

On implémente ce filtre avec la formule suivante en VHDL :

$$out = ((in0 + in1) + (in2 + in3) + 2 * in4 + (in5 + in6) + (in7 + in8))/10$$

Puissance totale : $Ptotale = 854.56\,mW$

Temps de réaction : $T = 93.629\,ns$ par pixel

Pour déterminer le débit d'images envisageable, prenons à titre d'exemple une image de taille 1920x1080.

Avec un bloc filtre le temps de traitement d'image est de :

$$93.629ns * 1920 * 1080 = 194ms$$

Ce qui fait un débit d'image d'environ 5 images par secondes. Pour pouvoir avoir un débit de 60fps, il faudrait donc utiliser 12 blocs filtres en parallèle.

En changeant la formule en ajoutant ou enlevant des parenthèses aux opérations, on peut forcer la synthèse RTL à paralléliser le filtre. On peut voir ci-dessous la version non parallélisée et la version parallélisée. On remarque que les résultats en simulation sont identiques dans les 2 cas, car l'outil de synthèse se charge déjà d'optimiser le code (le schémas RTL n'est pas représentatif de la réalité).

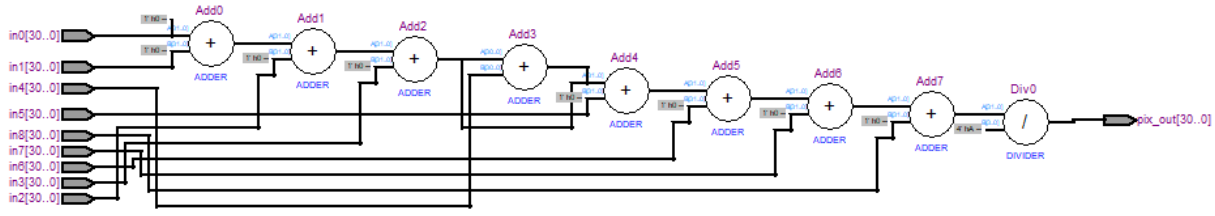


Figure 5- RTL non optimisé

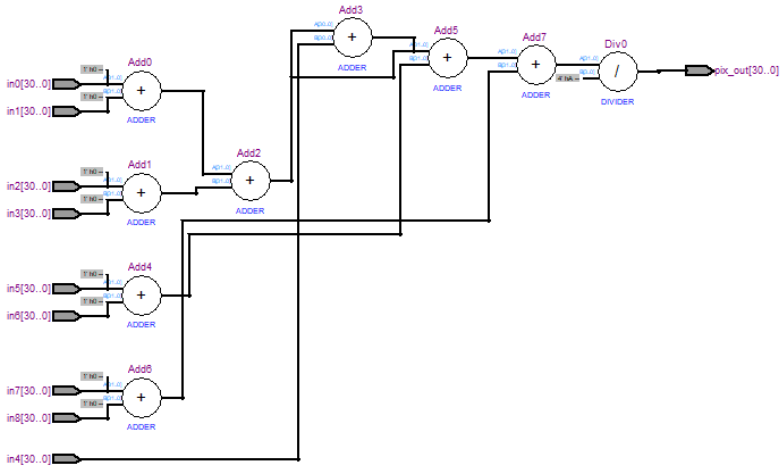


Figure 6- RTL optimisé

5 CONCLUSION

Lors d'un développement FPGA, il est nécessaire si l'on veut contrôler la consommation de notre application de prendre en compte certaines interactions.

Si l'objectif est de diminuer la puissance statique, il faut vérifier que le nombre de bits qu'on utilise soit toujours le plus bas possible.

Si l'objectif est de diminuer la puissance dynamique et que l'on n'a pas la main sur la tension d'alimentation, on peut diminuer la fréquence de fonctionnement. Il faut aussi vérifier que pour les opérateurs basiques on utilise bien les DSP, ce qui nous permet d'optimiser la surface, le temps de réponse, et la consommation.

Il peut aussi être intéressant de pipeliner les opérateurs pour réduire la consommation moyenne, mais il faut faire attention à la puissance instantanée qui peut vite augmenter.

Il n'est par contre pas nécessaire d'optimiser le RTL car l'outil de synthèse s'occupe déjà de faire les optimisations nécessaires.