Question 1

|  |  |  |  |
| --- | --- | --- | --- |
| **Changements** | **Performance HLS (latence cycles)** | **Performance HLS (% BRAM, % DSP, % FF, % LUT)** | **Performance exécution (image/sec)** |
| 1. Sobel filter HW départ | 452864553 | (1, ~0, 1, 6) | ~0.05 |
| 2. En ajoutant cache à une dimension | Min : 35295626  Max : 136996106 | (2, 1, 2, 7) | / |
| 3. Ajout de pipeline sur la boucle des colonnes | 64350761 | (2, 0, 2, 8) | ~0.5 |
| 4. Partitionnement d’une cache à deux dimension et refactoring du code de sobel\_operator et flatten\_off du pipeline | 4171584 | (2, 0, 1, 5) | ~25 |

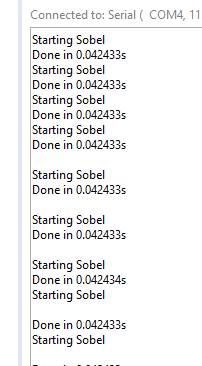
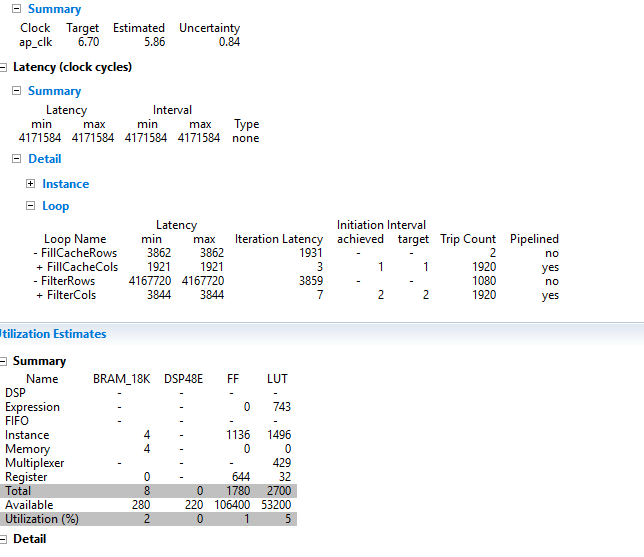
Explications de l’amélioration de performance :

Passage 1 à 2 : Lire de l’information d’un pointeur extérieur demande plus de cycles que de lire d’un tableau local. L’implémentation de la cache permet de limiter la lecture sur le pointeur à une seule fois par itération de la boucle intérieure et de faire cette lecture en rafales. L’opérateur de la boucle fait ces lectures sur la cache pour ses calculs.

Passage de 2 à 3 : Le pipeline demande plus de LUT en termes de ressources mais réduit la demande en DSP. Il a peu d’impact sur la latence mais le pipelinage permet d’augmenter le débit et d’améliorer ainsi le nombre d’images traitées par seconde.

Passage de 3 à 4 : La directive flatten\_off diminue les ressources consommées par le pipeline. Le code de l’opérateur Sobel a été changé pour utiliser une image en deux dimensions (la nouvelle cache) comme paramètre. Le partitionnement de la première dimension de la cache en plusieurs BRAM permet de paralléliser les accès lecture de la cache. Cela nous permet donc de réduire la latence, car il n’y a qu’un seul port de lecture par BRAM, mais nous faisons des accès à des BRAM différentes.

Screenshots du résultat final :



Question 2

1. 452864553 cycles \* 10 ns/cycle = ~ 4.5s. Ce qui nous donne ~0.22 images/sec, ce qui est sensiblement différent du 0.05 images/sec.
2. Min : 35295626 cycles \* 10 ns/cycle = ~ 0.35s, Max : 136996106 cycles \* 10 ns/cycle = ~ 1.3s. Nous obtenons donc Min : ~2.85 images/sec et Max : ~0.77 images/sec. Nous n’avons pas obtenu les résultats de l’implémentation sur la carte (on t’en avait parlé que nous n’allions en faire que trois parce que ça prend pas mal de temps par changement).
3. 64350761cycles \* 10 ns/cycle = ~ 0.64s. Ce qui nous donne environ 1.5 images/sec, ce qui est assez différent du 0.5 images/sec
4. 4171784 cycles \* 10 ns/cycle = ~ 0.04s.

|  |  |  |  |
| --- | --- | --- | --- |
| **Changements** | **Performance HLS (latence cycles)** | **Performance HLS (% BRAM, % DSP, % FF, % LUT)** | **Performance exécution (image/sec)** |
| 1. Sobel filter HW départ | 452864553 | (1, ~0, 1, 6) | ~0.05 |
| 2. En ajoutant cache à une dimension | Min : 35295626  Max : 136996106 | (2, 1, 2, 7) | / |
| 3. Ajout de pipeline sur la boucle des colonnes | 64350761 | (2, 0, 2, 8) | ~0.5 |
| 4. Partitionnement d’une cache à deux dimension et refactoring du code de sobel\_operator et flatten\_off du pipeline | 4171584 | (2, 0, 1, 5) | ~25 |