1 Analyse

1.1 Virgule fixe

Comme les calculs sont réalisés sur FPGA, il est plus adapté d'utiliser des nombres en virgule fixe. Étant donné que les DSP sont sur 18 bits, les nombres seront eux aussi sur 18 bits avec :

- Un bit de signe
- Deux bits avant la virgule (pour écrire des nombres entre 3 et -4)
- Quinze bits après la virgule ce qui permet d'avoir une résolution de 3.05×10^{-5}

1.2 Organisation

La base du calcul (l'itérateur) sera réalisé en premier. Il s'agit d'un bloc VHDL capable d'effectuer le calcul

$$z_{k+1} = z_k^2 + C$$

Tout en indiquant si le rayon limite est atteint et le nombre d'itérations en sortie. Si l'entrée `done_in` est activée, alors le nombre d'itérations en sortie est le même que le nombre d'itérations en entrée.

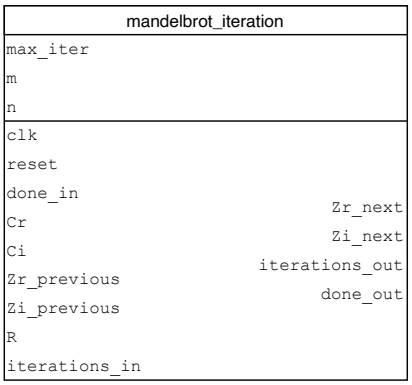

FIGURE 1 – Bloc `mandelbrot_iteration`

Ce bloc sera ensuite réutilisé dans deux versions du programme

1. Bouclage sur lui-même (loop) pour calculer un pixel à la fois
2. Pipeline, 100 itérateurs (le nombre maximal d'itérations) sont reliés les uns après les autres ce qui permet de calculer un pixel par coup d'horloge

1.2.1 Bouclage

Les sorties de l'itérateur sont reliées sur les entrées à l'exception du premier itérateur (qui prend les entrées pour le point à calculer) et du dernier (qui fourni le nombre d'itérations en sortie)

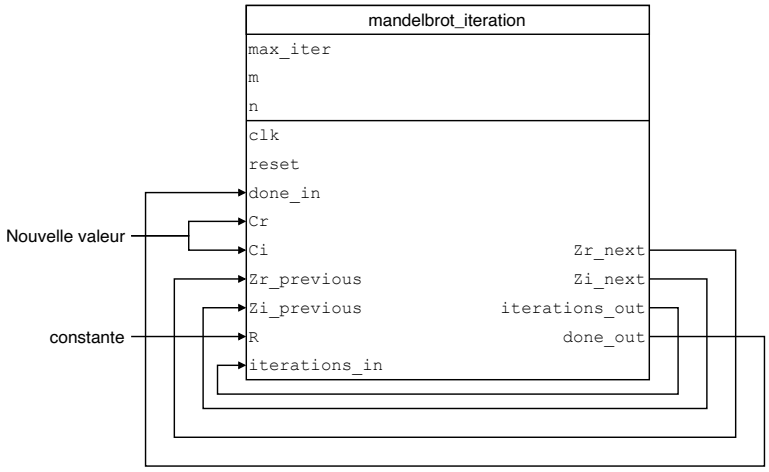

FIGURE 2 – Bloc `mandelbrot_loop`

1.2.2 Pipeline

Les itérateurs sont reliés à la suite les uns des autres, les entrées `Cr` et `Ci` sont propagées aux même rythme que le pipeline (registres à la suite)

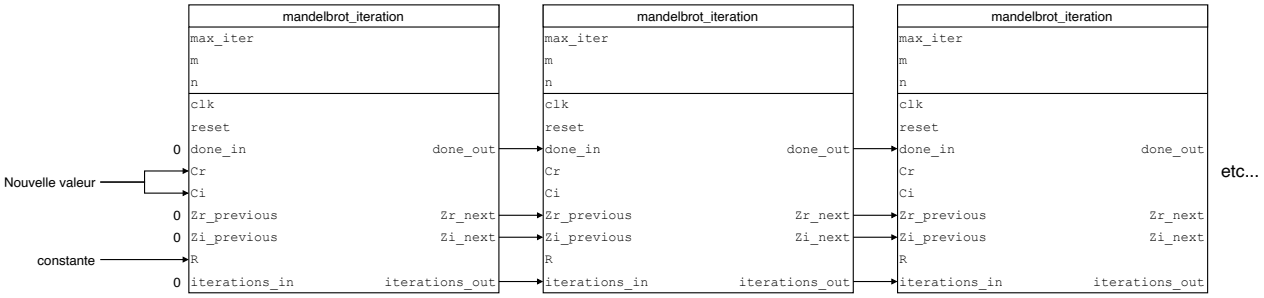

FIGURE 3 – Bloc `mandelbrot_pipeline`

1.3 Calculs sur Python

Avant d'implémenter la fractale sur FPGA, il est souhaitable d'avoir un modèle complet pour comparer et débugger le système. Le modèle sera réalisé dans Python en utilisant le package `fixedpoint` [1]. De cette façon il est possible d'effectuer les calculs en virgule flottante (pour avoir la "vraie" fractale) puis les mêmes calculs en virgule fixe afin de comparer de pouvoir créer un banc de test VHDL.

1. `https://pypi.org/project/fixedpoint/`

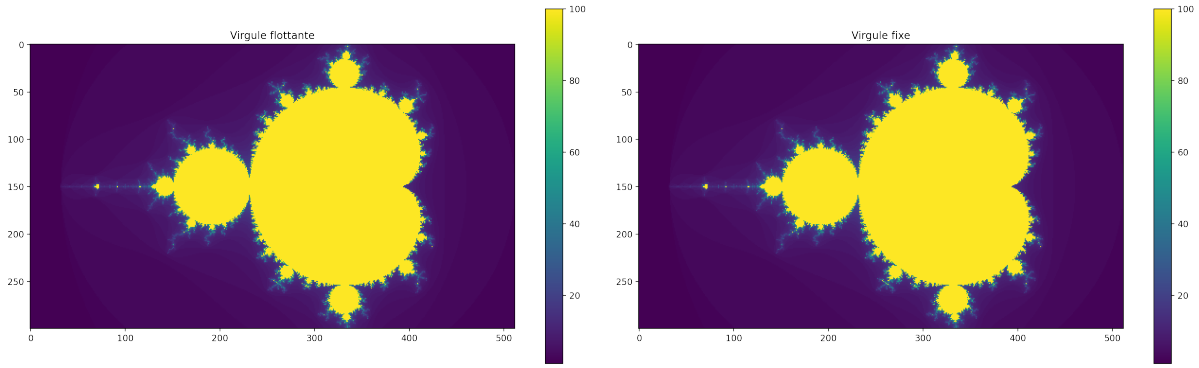

FIGURE 4 – Fractales avec virgule fixe et virgule flottante

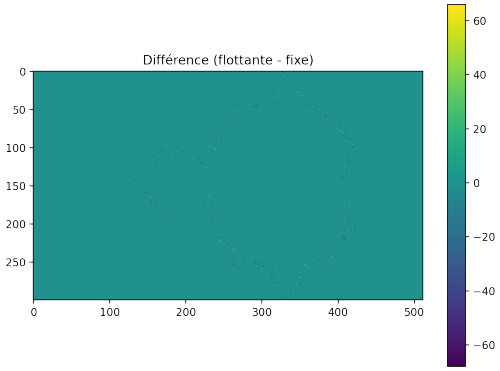

FIGURE 5 – Différence entre les deux résultats

On remarque qu'il existe une différence entre les deux approches (à cause des arrondis) mais elle est négligeable dans notre application car elle n'impacte pas le rendu visuel.

1.4 Couleurs

Afin de déterminer les couleurs de chaque pixel en fonction du nombre d'itérations, on utilise une interpolation sur des points donnés pour chaque couleur (rouge, vert, bleu). Il existe plusieurs choix, mais le celui donné par cette réponse [2] donne un bon résultat

2. `https://stackoverflow.com/a/25816111/8418125`

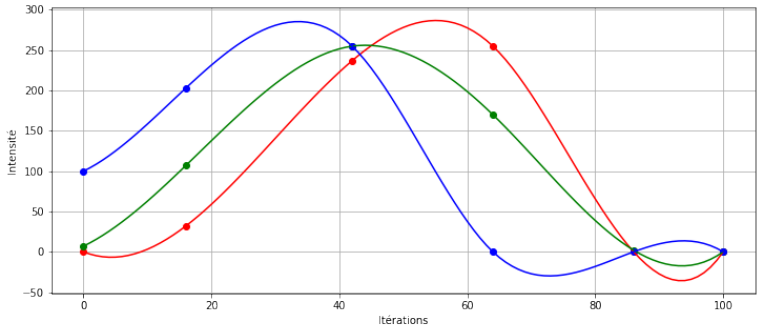

FIGURE 6 – Interpolation cubique sur les couleurs

Après ajustage (pour borner les intensités entre 0 et 255), on obtient la répartition de couleurs suivante en fonction du nombre d'itérations :

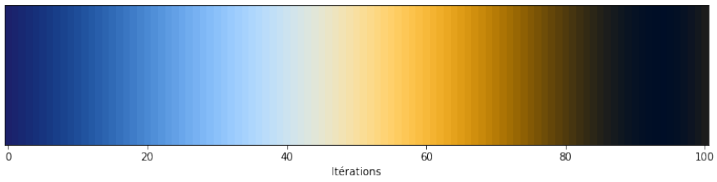

FIGURE 7 – LUT pour la couleur de chaque pixel en fonction des itérations

La LUT est auto-générée dans un package VHDL.