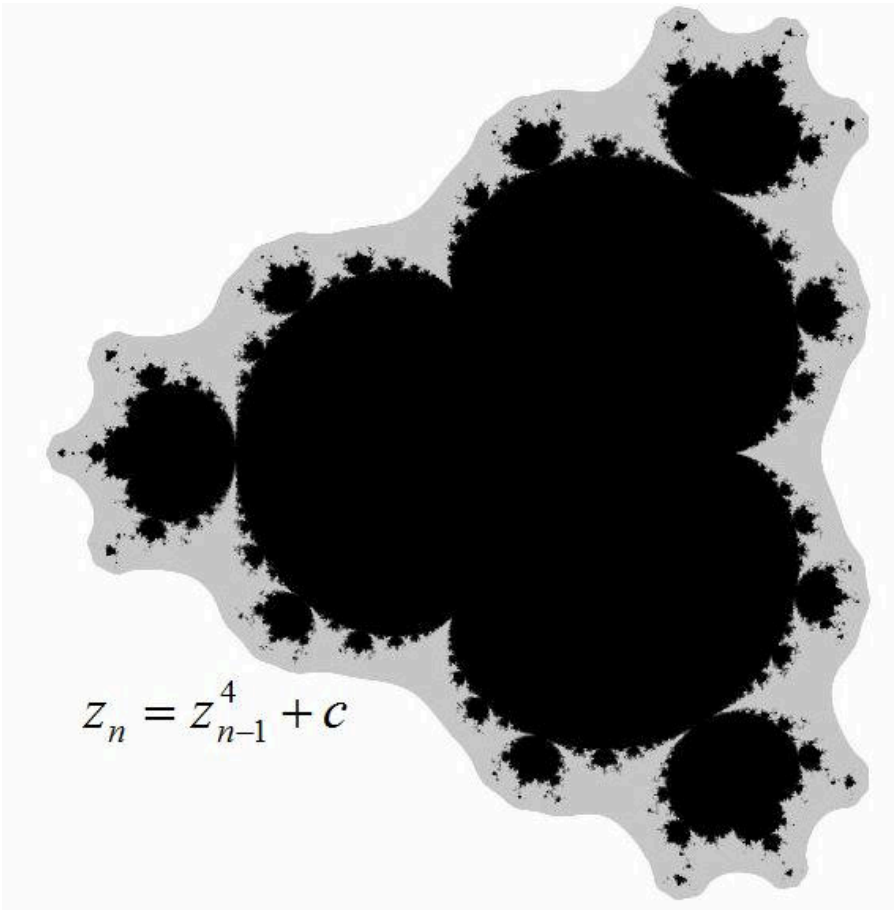
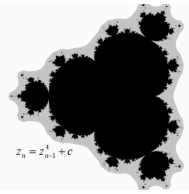
Rapport LPSC

Générateur de fractale de Mendelbrot,
$Z^4 + C$

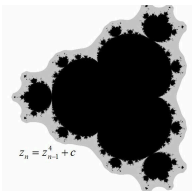


HES-SO MSE
Electrical Engineering - Weak current
LPSC - Fabien Vannel
Semestre de Printemps 2025 -

Victor Cochet
09.06.2025 - V1.0 - final



Contents



1 Introduction

1.1 But

Dans le cadre d'un cours du Master HES-SO, il nous a été donné comme tâche de réaliser un générateur de fractale de Mandelbrot en utilisant un circuit électrique basé sur un FPGA de chez Xilinx. Nous devons utiliser les propriétés de calcul de la plateforme pour réaliser un calcul itératif qui peut beaucoup de ressources et être passablement complexe dans un système exécutant du code procédural. La possibilité d'exécuter des actions en parallèle constitue l'avantage principal du FPGA et va nous permettre d'effectuer très rapidement les calculs.

1.2 Données du projet

J'ai dû implémenter la fractale correspondant à la formule suivante :

$$Z_{\mathrm{n+1}} = Z^4 + C \tag{1}$$

Avec C correspondant à la coordonnée du point sur lequel on effectue le calcul. C'est une opération itérative qui va nous permettre de déterminer une valeur d'itération correspondante à la couleur associée à cette coordonnée. De plus, mon système doit être basé sur une architecture parallèle. Pour ce faire, j'ai décider d'utilisé principalement des DSP48E1 et le VHDL. On développe l'Equation 1 en utilisant les valeurs littérales suivantes :

$$Z = Z_{\mathrm{Re}} + iZ_{\mathrm{Im}} \tag{2}$$

$$C = C_{\mathrm{Re}} + iC_{\mathrm{Im}} \tag{3}$$

On peut développer les parties réels et imaginaires suivantes :

$$Z_{\mathrm{Re}_{\mathrm{n+1}}} = Z_{\mathrm{Re}}^4 - 6(Z_{\mathrm{Re}}Z_{\mathrm{Im}})^2 + Z_{\mathrm{Im}}^4 + C_{\mathrm{Re}} \tag{4}$$

$$Z_{\mathrm{Im}_{\mathrm{n+1}}} = 4\big(\big(Z_{\mathrm{Re}}^3 Z_{\mathrm{Im}}\big) - \big(Z_{\mathrm{Re}} Z_{\mathrm{Im}}^3\big)\big) + C_{\mathrm{Im}} \tag{5}$$

Avec une petite transformation, on obtient :

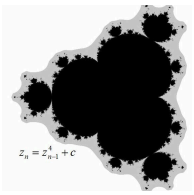
$$Z_{\mathrm{n+1}_{\mathrm{Re}}} = ((Z_{\mathrm{Re}}^2 - Z_{\mathrm{Im}}^2) - (2Z_{\mathrm{Re}}Z_{\mathrm{Im}}))((Z_{\mathrm{Re}}^2 - Z_{\mathrm{Im}}^2) + (2Z_{\mathrm{Re}}Z_{\mathrm{Im}})) + C_{\mathrm{Re}} \tag{6}$$

$$Z_{\mathrm{n+1}_{\mathrm{Im}}} = 4(Z_{\mathrm{Re}}Z_{\mathrm{Im}}(Z_{\mathrm{Re}}^2 - Z_{\mathrm{Im}}^2)) + C_{\mathrm{Im}} \tag{7}$$

Le but de ce développement est d'identifier un maximum d'éléments communs, ce qui me permet de limiter le nombre d'instances de calculateurs complexes nécessaires sur le FPGA et donc d'augmenter potentiellement le nombre de calculateurs pouvant être mis en parallèle. En décomposant les termes qui sont des identités remarquables, on peut réduire le nombre de DSP à utiliser :

$$Z_{\mathrm{Re}}^2 - Z_{\mathrm{Im}}^2 = (Z_{\mathrm{Re}} - Z_{\mathrm{Im}})(Z_{\mathrm{Re}} + Z_{\mathrm{Im}}) \tag{8}$$

Avec l'Equation 8, on peut économiser un DSP.



2 Architecture

Mon architecture est constituée d'un bloc calculateur d'itération qui fait les opérations mentionnées précédemment. J'ai opté pour une solution tenant dans une mémoire BRAM dans un premier temps, ce qui m'a conduit à utiliser une image qui fait 220x220 pixels centrée sur l'écran. Cela m'a permis dans un premier temps de pouvoir avoir toute l'image en mémoire et dans un deuxième temps, d'avoir assez de mémoire d'avance pour être sûre que les itérations calculées soient constamment en avance sur l'affichage.

2.1 Implémentation

Pour réaliser le projet, on nous a donné un canevas dans lequel nous devions implémenter notre architecture. Dans mon cas, j'ai créé un bloc VHDL contenant mon calculateur et je l'ai intégré en tant qu'entité dans le fichier « scalp_user_design ». Cela m'a donné l'implémentation suivante :

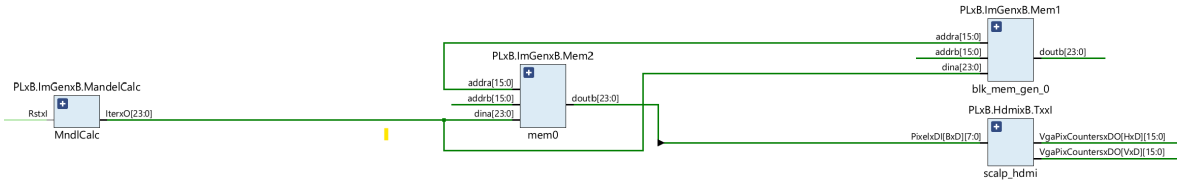


Figure 1 - Schéma d'implémentation du générateur de fractale

La Figure 1 montre l'implémentation résultante du code VHDL. Le bloc MandelCalc calcule pour chaque pixel de l'image la valeur d'itération sur 24 bits. Mem1 et Mem2 sont des mémoires contenant les valeurs de l'image actuelle et celle qui est en train d'être calculée. Le bloc HdmixB sert à afficher sur l'écran, celui-ci nous a été donné. Lorsque le bloc MandelCalc a fini de calculer toutes les valeurs pour chaque pixel de l'image, il signal à travers un buffer au bloc HdmixB que l'on est prêt à afficher l'image en mémoire, simultanément, le calculateur recommence à calculer et à stocker les résultats dans l'autre mémoire. Par ce mécanisme, on écrit dans une mémoire l'image qui doit être générée à la prochaine frame. Les deux mémoires servent à traverser les domaines d'horloge, le calculateur possède une horloge à 125 MHz alors que le bloc HdmixB fonctionne à 48 MHz. Le but final était de pouvoir avoir plusieurs blocs MandelCalc qui calculent en parallèle certaine partie de l'image de manière à être assez rapide pour que l'HdmixB puisse rafraîchir l'image à partir de la mémoire.

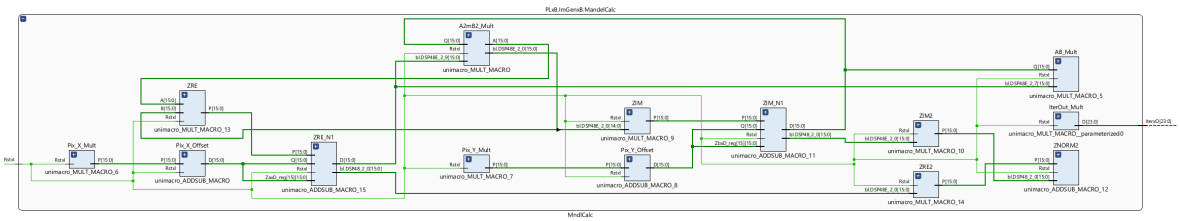
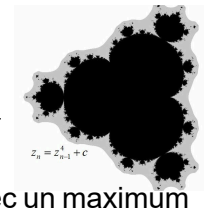


Figure 2 - Schéma d'implémentation du calculateur d'itération



La Figure 2 montre la structure interne du calculateur. Il a été implémenté avec un maximum de DSP. Il y en a 14 au total avec 5 qui sont utilisés en additionneur et 9 comme multiplicateurs. La plupart sont utilisés pour effectuer l'opération complexe, les autres sont là principalement pour convertir la valeur du pixel en coordonnée dans le plan complexe et un autre pour multiplier la valeur d'itération de sortie de manière à avoir directement une couleur en sortie sur 24 bits. L'équation qui permet de convertir le pixel en coordonnée a la forme suivante :

$$C_x = (\mathrm{Pix}_{\mathrm{act}} - \mathrm{Pix}_{\mathrm{off}})\left(\frac{\mathrm{R}}{\mathrm{Nbr}_{\mathrm{Pix}}}\right) + \mathrm{Offset} \tag{9}$$

Avec C_x la coordonnée finale que l'on souhaite obtenir, $\mathrm{Pix}_{\mathrm{act}}$ la valeur du pixel pour lequel on fait le calcul, $\mathrm{Pix}_{\mathrm{off}}$ la valeur du pixel de départ, R qui est l'amplitude que l'on veut couvrir sur l'axe des complexes, $\mathrm{Nbr}_{\mathrm{Pix}}$ le nombre de pixel sur notre image et finalement, Offset le début de notre coordonnée. Pour avoir un système d'axe qui corresponde à l'affichage correct de la fractale sur l'écran j'ai choisi d'avoir mon axe des réels allant de $[-1.5; 1]$ et mon axe imaginaire allant de $[-1.5; 1.5]$ ce qui nous donne en utilisant l'équation Equation 9 :

$$C_{\mathrm{Re}} = (\mathrm{Pix}_{\mathrm{act}} - \mathrm{Pix}_{\mathrm{off}})0.01136 - 1.5 \tag{10}$$

$$C_{\mathrm{Im}} = (\mathrm{Pix}_{\mathrm{act}} - \mathrm{Pix}_{\mathrm{off}})0.01363 - 1.5 \tag{11}$$

2.2 Bilan d'utilisation

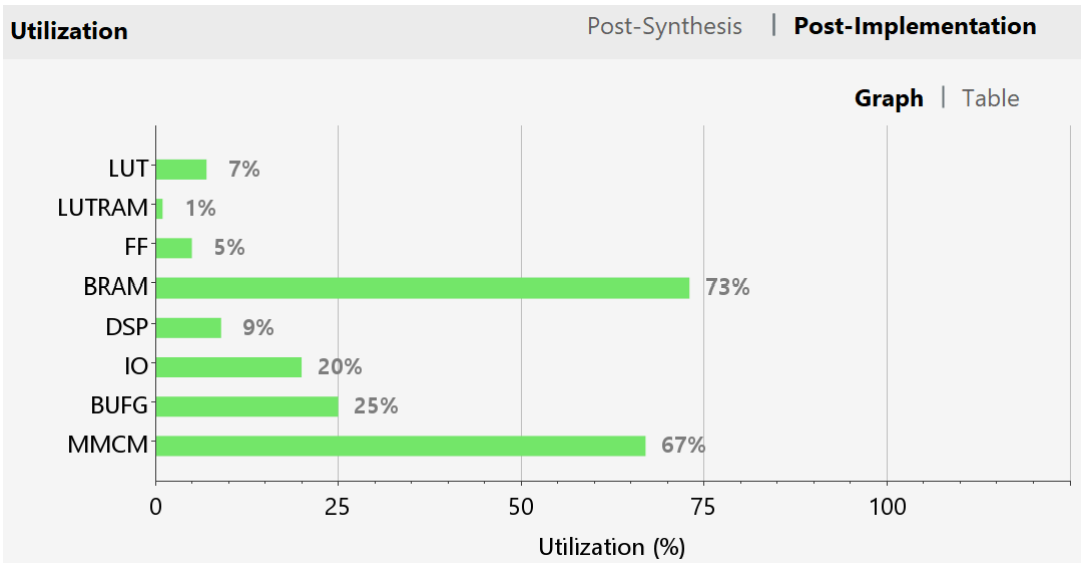
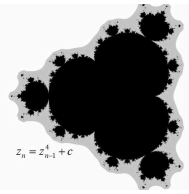


Figure 3 - Bilan d'implémentation

La Figure 3 montre le bilan d'utilisation du hardware avec un calculateur ainsi que deux BRAM pouvant contenir chacune une image de 220x220 pixels. On voit que j'ai utilisé 9% des DSP disponibles ainsi que 73% de la mémoire. C'est avantageux dans l'optique où je comptais instancier plusieurs blocs de calcul en parallèle. Cela veut dire que j'aurais pu créer au maximum 10 blocs de calcul avant que le FPGA n'ait plus de DSP disponible.



3 Conclusion

3.1 Résultats

3.1.1 Performance du calculateur

Le calculateur implémenté est capable de produire le résultat de l'Equation 1 en 1 cycle d'horloge, le test des conditions d'arrêt prend un cycle d'horloge en plus ce qui fait que notre calculateur prend dans le pire des cas, 101 cycles d'horloges pour émettre une valeur de couleur associée au pixel. Cela correspond à une fréquence de calcul allant de 62.5 MHz à environ 125MHz.

3.1.2 Etat du projet

J'ai pu implémenter le générateur avec le schéma de la Figure 1. La fractale de Mandelbrot apparaît comme attendu, cependant l'affichage a un problème que je n'ai pas résolu. La figure qui apparaît se dépalace en diagonale sur l'écran. Comme la fonction d'agrandissement n'a pas encore été implémentée, cela suggère que la mémoire lue ne commence pas à l'adresse initiale attendue par l'écran. En d'autres termes, je n'ai pas encore réussi à faire en sorte que l'écran commence la lecture de la BRAM à la bonne adresse lorsque l'affichage débute.