GLEYZES Vincent
MAYOR-SERRA Anthony
Groupe A2
2015-2016

Projet C++ :
Etude d'un système processeur, GPU et accélérateurs matériels (VHDL) appliqué au calcul de fractales
PR214

Professeur Encadrant : Bertrand Le Gal

Table des matières

I-Introduction

L'ensemble de Mandelbrot constitue un défi mathématique et informatique depuis sa découverte par Benoît Mandelbrot. D'une simple définition de convergence, il s'avère l'application d'un calcul générant des formes géométriques très complexes, et surtout, qui peut rapidement s'avérer très consommatrice en ressources numériques.

Ce rapport aborde la problématique de l'ensemble de Mandelbrot en proposant plusieurs types d'implémentation de l'algorithme pouvant le générer.

Une présentation générale des données de l'étude sera d'abord faite, puis les différentes implémentations seront abordées, et enfin un comparatif des méthodes la conclura.

II-Présentation de l'étude : l'ensemble de Mandelbrot

II-1) Ensemble de Mandelbrot – Définitions, mise en pratique

L'ensemble de Mandelbrot est régit par l'équation :

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

Si lors du calcul récurrent des termes de la suite Zn, ceux-ci gardent un module ne divergeant pas lorsque n→ +∞, alors ils appartiennent à l'ensemble de Mandelbrot.

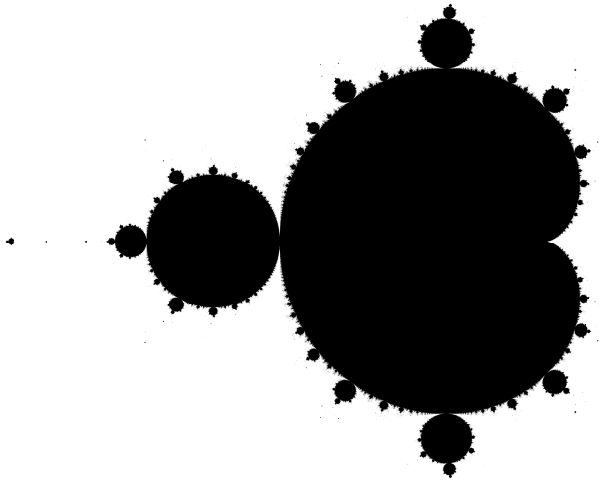

FIGURE 1 : ENSEMBLE DE MANDELBROT

En pratique, il est impossible de générer une infinité d'itérations sur une infinité de coordonnées. L'affichage de cet ensemble correspond donc :

- d'une part, à la discrétisation du domaine cartésien étudié
- d'autre part, au choix arbitraire d'une limite d'itération pour laquelle la suite sera considérée convergente.

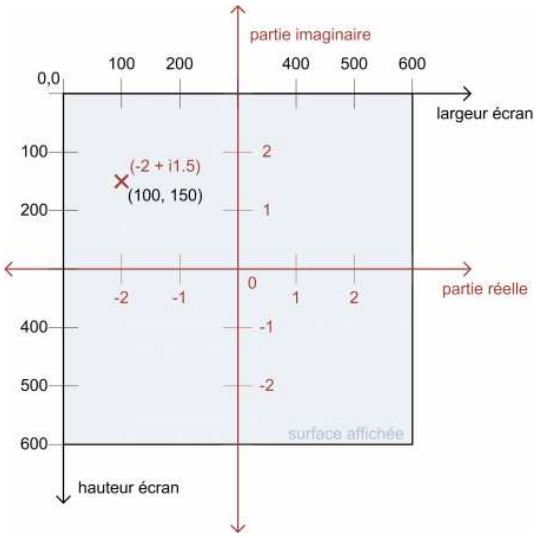

FIGURE 2 : DISCRETISATION DU DOMAINE ETUDIE

La discrétisation du domaine cartésien étudié est naturelle: elle est en correspondance avec la discrétisation générée par les pixels d'un écran. Ainsi, chaque pixel est projeté sur le plan et génère un couple de coordonnées séparés par un pas.

Le choix de la limite d'itération, en revanche, peut-être fortement liée à la vitesse de calcul escomptée. Ainsi, il peut dépendre :

- de la résolution de l'écran, le nombre de pixels définissant en lui-même le temps de rendu total
- du nombre d'itération maximal, qui peut considérablement augmenter le temps de rendu des coordonnées qui tendent à converger
- de la fréquence d'utilisation du dispositif numérique, ainsi que de son architecture
- de la taille du domaine d'affichage (plus le domaine est de faible taille, plus le nombre d'itération doit être élevé pour un affichage correct)

Finalement, un point complexe sera considéré comme divergent si son module est supérieur à deux durant une itération.

II-2) Algorithme

Quel que soit la méthode d'implémentation utilisée, le calcul de l'ensemble de Mandelbrot correspond à un algorithme commun (non optimisé) :

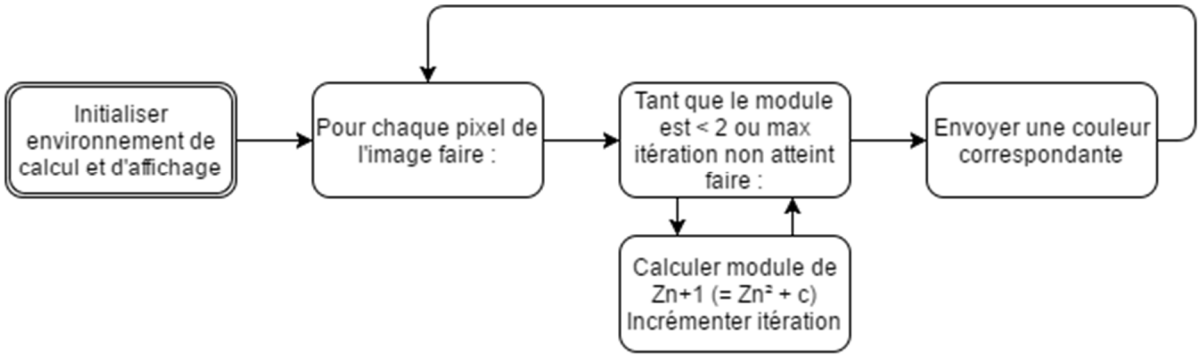

FIGURE 3 : ALGORITHME DE CALCUL DE L'ENSEMBLE DE MANDELBROT

A chaque $Z_n=X_n+jY_n$, il faut déterminer $Z_{n+1} = X_{n+1} + j\ Y_{n+1}$, avec :

$$X_{n+1}=X_0+X_n^2-Y_n^2 \quad \text{et} \quad Y_{n+1}=Y_0+2*X_n*Y_n$$

Le défi de cet algorithme n'est donc pas sa complexité, mais sa redondance. Il correspond à un temps de calcul en O(résolution en X * résolution en Y * nombre maximal d'itérations *opération mathématique).

L'opération mathématique effectuée utilisant des multiplications, il est aisé d'affirmer que le temps de calcul peut rapidement s'avérer pharamineux.

3) Réduction du temps de calcul

Une limite inhérente à cette méthode de calcul est la récursivité. Ainsi, il est impossible de calcul le terme suivant Zn+1 sans le terme courant Zn, rendant cette étape incompressible.

En revanche, la méthode de calcul est parfaitement identique pour chaque pixel de l'image à générer. Il est donc envisageable de paralléliser ce type de calcul.

Dans les parties suivantes du rapport, trois approches seront abordées pour faire une comparaison d'implémentation : implantation purement logicielle, implantation sur processeur plasma (implantation hydride VHDL), et implantation en VHDL pure.

Une première étape immédiate est d'effectuer les tests de convergences sur le carré du module, qui doit donc être inférieur à 4. Toutes les implémentations utiliseront cette optimisation qui permet d'enlever rapidement un calcul de racine carrée.

4) Paramètres, format des données, précision

Afin de donner à la comparaison des implantations un certain réalisme, certains paramètres d'implémentation et de test seront identiques, notamment :

- Résolution : 640x480
- Maximum d'itérations : 4096
- Pour les implantations VHDL : 4 cœurs de calcul
- Le domaine maximal d'affichage sera considéré comme étant X[-2, 1], Y[-1, 1] pour le choix du format des données

Le format de calcul des données sur VHDL sera en virgule fixe, car plus simple à implémenter.

Il sera composé d'un bit de signe, de 3 bits de partie entière, et de 28 bits de partie fractionnaire. Ainsi, le plus petit pas représentable sera 2^{-28}, soit $3{,}725.10^{-9}$, au-delà duquel le dispositif ne pourra plus assurer sa fonction d'affichage.

Ce format de données à l'avantage d'être utilisable directement par le processeur plasma et assure ainsi la précision la plus élevée possible avec l'implémentation en virgule fixe.

II-Implémentation logicielle

II- 1) Fonctionnement

```
int convergence(double x, double y)
{
    int i;
    double uRe=x, uIm=y,a,b;

    for(i=0;i<ITER; i++)
    {
        a = x + uRe*uRe - uIm*uIm; //partie réelle
        b = y + 2*uRe*uIm; //partie imaginaire
        uRe = a;
        uIm = b;

        if ((uRe*uRe + uIm*uIm) >= 4)
            return i;
    }
    return 0;
}
```

L'implémentation logicielle est une version simple de l'application directe de la formule de calcul de $|Zn|^2$. La valeur du couple de coordonnées (Xn+1,Yn+1) est stocké temporairement puis le module carré est déterminé.

Chaque nombre d'itérations est ainsi renvoyé, et cette fonction est appelée pour chaque pixel de l'écran.

II-2) Performances

La génération d'une image 640x480 prend environ 3 secondes, dont environ 0.15 secondes d'écriture du fichier. Le calcul d'une telle fractale représente donc un temps non négligeable.

III-Implémentation en VHDL

III-1) Structure

L'implantation en VHDL pure exige une gestion totale de tous les processus de calcul de la fractale de Mandelbrot. Chaque fonction explicitée dans l'algorithme de l'illustration 3 doit donc être assurée par un module VHDL.

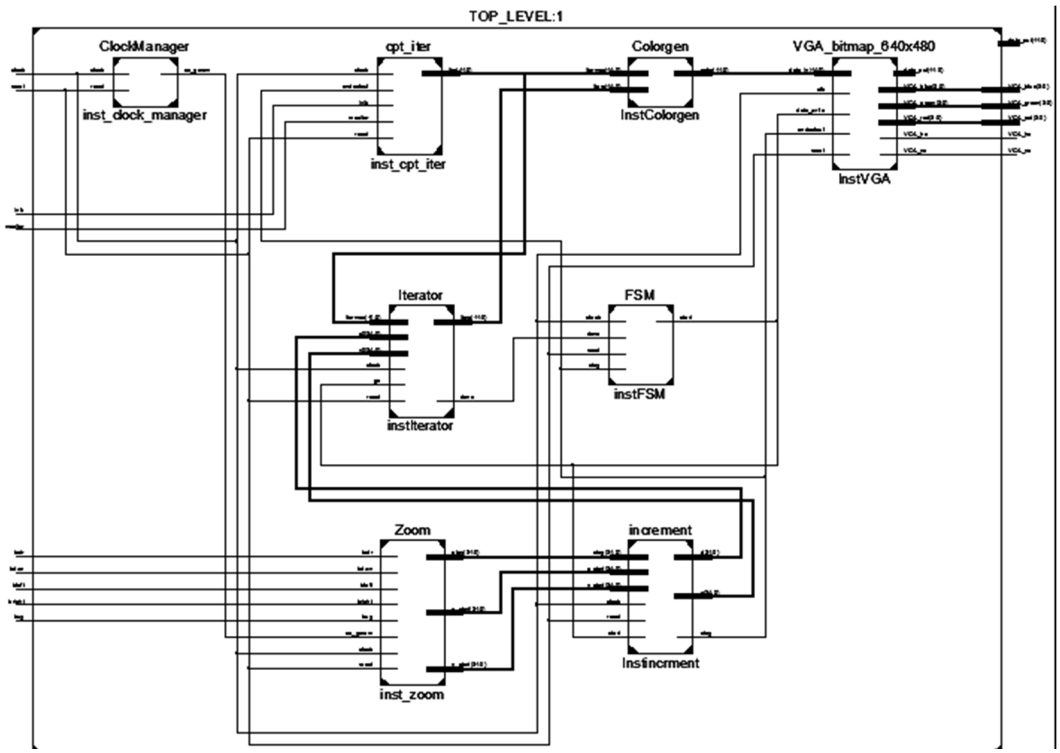

FIGURE 4 : SCHEMA BLOC DE L'ARCHITECTURE EN VHDL

L'initialisation des valeurs est réalisée en interne dans chaque module utilisant une variable susceptible d'être recalculée.

Les modules primordiaux à l'affichage de l'ensemble sont :

- VGA_bitmap_640_480, mémoire RAM contenant les données à envoyer au dispositif d'affichage. Le module contient sa propre gestion d'adresse, la RAM est parcourue cycliquement, et dans l'ordre.
- Colorgen, génère une couleur codée sur 12 bits (format par défaut pour limiter la place prise par la RAM) en fonction de la valeur d'itération reçue.
- Iterator, calcule chaque $|Zn|^2$ d'un complexe passé en entrée (sous le format virgule fixe, X et Y du complexe X + j Y) et envoie le nombre d'itérations au Colorgen.
- Increment, parcours en interne via des compteurs l'ensemble des pixels et envoient leurs coordonnées cartésiennes correspondantes à Iterator.
- FSM, séquence les étapes de calcul et l'envoi au VGA du résultat.

Des modules facultatifs ont été ajoutés pour assurer des fonctionnalités supplémentaires :

- Zoom, permet de modifier les coordonnées de départ de l'ensemble affiché, ainsi que le pas d'affichage (et donc le zoom).
- cpt_iter, permet d'incrémenter à chaque fin de rendu le nombre maximal d'itérations de convergence afin d'avoir un raffinement progressif de la fractale.
- ClockManager, filtre les boutons afin qu'ils n'interagissent avec un paramètre donné que toutes les demi-secondes (principe de Clock Enable).

III-2) Interface

L'interface d'utilisation de l'implémentation VHDL sur Nexys 4 est la suivante :

- b_left, b_right, b_up, b_down : déplacement dans l'ensemble de Mandelbrot
- b_ctr + b_up : zoom
- b_ctr + b_up : dézoom
- Switch 0 : Reset (réinitialise coordonnées de départ, max d'itérations et zoom)
- Switch 1 : inib, active ou désactive l'incrémentation régulière du max d'itérations
- Switch 2 : maxiter, force le max d'itérations à valoir 4096 (prioritaire sur inib)

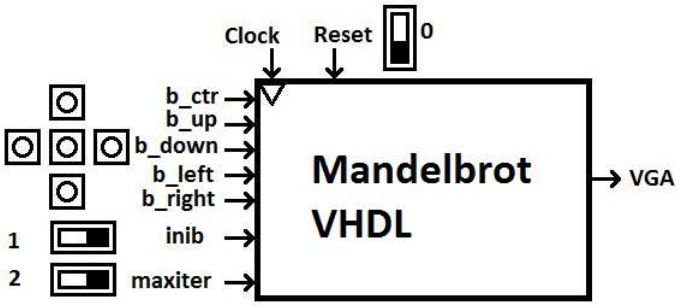

FIGURE 5 : SCHEMA DES ENTREES SORTIES

III-3) Détails de l'implémentation du calcul

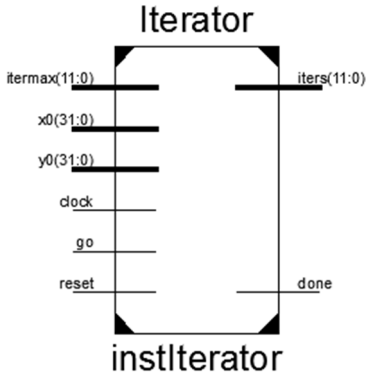

FIGURE 6 : SCHEMA BLOC DU MODULE ITERATOR

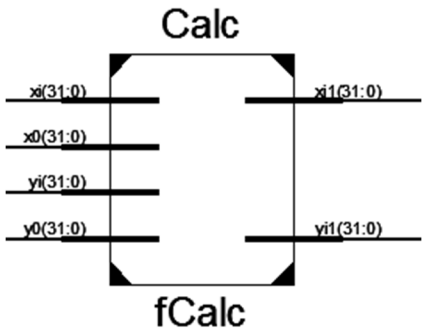

FIGURE 7 : SCHEMA BLOC DU MODULE CALC

Iterator est un module synchrone. Initialement, le nombre d'itération qu'il stocke vaut 0. Sur réception depuis la FSM de l'ordre de départ de calcul (go), il utilise les coordonnées x0 et y0, correspondant au complexe c dans :

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

Pour générer la suite des Zn. Il utilise le module Calc à cet effet qui est dédié au calcul est coordonnées de chaque Zn+1 depuis celles de Zn et c.

Il fait ainsi intervenir le module combinatoire Calc pour chaque itération, jusqu'à ce que la condition de convergence ne soit plus respectée ou que le max d'itérations ait été atteint. Iterator envoi enfin le nombre d'itérations réalisé à Colorgen ainsi qu'un signal de fin (done) à FSM pour signaler qu'il a terminé le calcul d'un pixel.

La valeur d'itération reçue par Colorgen sert d'adresse pour récupérer l'une des couleurs stockée par le module

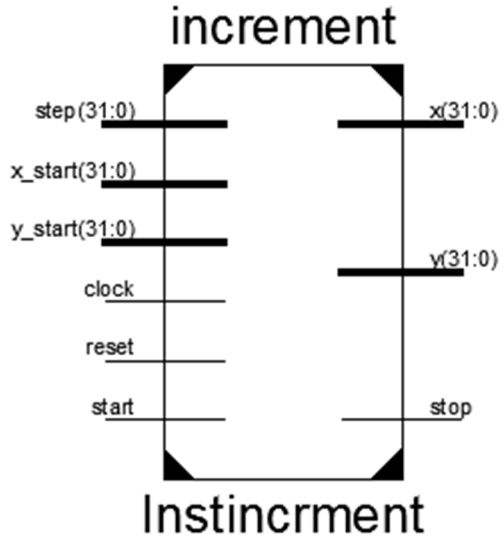

FIGURE 8 : SCHEMA BLOC DU MODULE INCREMENT

Increment s'occupe quant à lui des coordonnées associées à chaque pixel. Il récupère initialement la valeur des coordonnées de départ de l'ensemble à tracer ainsi que la valeur du pas (depuis Zoom), et détermine ainsi les coordonnées de chaque pixel (la résolution étant définie dans des constantes) sur cadencement de la FSM (start). Lorsque l'intégralité de l'image est constituée, il génère un signal de fin de rendu (stop) destiné à la FSM et VGA_bitmap_640_480.

III-4) Utilisation de la virgule fixe

La détermination de $|Zn|^2$ nécessite l'implémentation d'additions et de multiplications en virgule fixe. Si le VHDL permet d'utiliser l'addition directement depuis des SIGNED (en supposant qu'il n'y ai pas de débordement de capacité), la multiplication exige une implémentation particulière dépendante du format virgule fixe employé. Ainsi la multiplication de deux nombre au format F(4,28) va générer un nombre au format F(8,56). Il faut donc tronquer le résultat, pour une inévitable génération d'imprécision de l'ordre de 2^28.

```
function mult(A : SIGNED; B : SIGNED; QF : integer) return SIGNED is
   CONSTANT DMAX_R : integer := A'LENGTH + B'LENGTH;
   CONSTANT PHI : integer := A'LENGTH - QF;
   VARIABLE r  : SIGNED(DMAX_R-1 DOWNTO 0);

begin
   r := A*B;
   return r(DMAX_R - PHI - 1 downto QF);
end mult;
```

La fonction de multiplication utilise donc la multiplication directe de SIGNED, puis utilise :

- DMAX_R, la taille totale du résultat généré
- A'LENGTH, B'LENGTH, taille d'une donnée
- QF, taille de la partie fractionnaire d'une donnée
- PHI, la taille de la partie entière du format utilisé (taille d'une donnée – taille partie fractionnaire)

Elle génère donc une donnée au format F(PHI+PHI,QF+QF), et tronque donc en amont 1.PHI, et en aval 1.QF.

III-5) Parallélisation et performance

L'intérêt principal d'une approche VHDL est qu'elle n'a de limite que les ressources disponibles sur le FPGA. Ainsi, la mise en parallèle de plusieurs unités de calculs est possible, la détermination du nombre d'itération étant le point chaud du calcul de la fractale. Ainsi, il suffit d'instancier plusieurs modules Iterator, et de récupérer leur résultat au fur et à mesure :

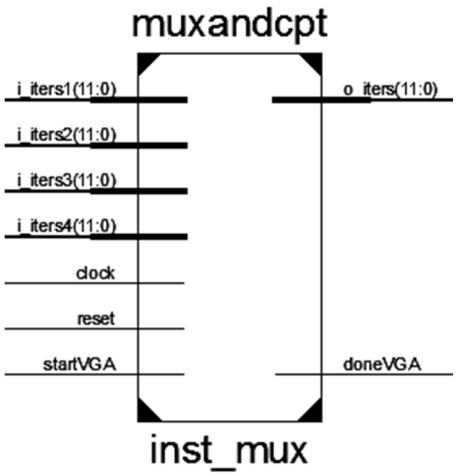

FIGURE 9 : SCHEMA BLOC DU MODULE MUXANDCPT

Le comportement de la FSM a été changé : elle attend que tous les Iterator aient finis leurs calculs avant d'envoyer l'ordre d'écriture des pixels. Ainsi, le temps de calcul se résume au pixel ayant le plus grand nombre final d'itération, plutôt que la somme de chacun d'entre eux.

L'ordre d'écriture est représenté par startVGA. Lorsque tous les pixels ont été écris en série dans VGA_bitmap_640_480, inst_mux envoi à la FSM un signal de fin d'écriture.

L'écriture en série des pixels étant dissociée de l'atteinte du résultat de tous les pixels calculés en parallèle, le dispositif génère une perte de temps, mais celle-ci est minime car ne faisant pas partie du point chaud.

Le comportement du reste de la description VHDL est peu changée. Le calcul des coordonnées d'un pixel reste le même, les pixels suivant pour la parallélisation étant déduit du premier et d'un multiple du pas.

Temps de calcul d'une image 640x480 à 4096 itérations maximum :

Nombre d'Iterator	1	2	4	8
Temps de calcul(s)	4	2.5	1.5	<1

Le gain de temps de calcul est fort pour une consommation en ressources VHDL faible (de 2 à 4 % des LUT de la Nexys 4 selon les versions). Il est donc possible d'imaginer une accélération plus élevée.

Une autre piste serait d'utiliser une fenêtre glissante pour limiter le ralentissement généré par le pixel ayant l'itération la plus élevée.

IV-Implémentation sur le CPU Plasma

L'objectif dans cette partie est d'implémenter le calcul de la fractale de Mandelbrot sur le CPU plasma qui est un CPU codé en VHDL et est donc reconfigurable. L'idée étant ensuite de multiplier le nombre de processeur implémentés sur la carte afin d'accélérer le calcul de chaque images.

L'avantage de cette plateforme est que le code de l'algorithme et des calculs est réalisé en C et est envoyé lors de la synthèse du processeur dans la mémoire boot qui charge le programme au démarrage de la carte. On obtient ainsi un compromis entre facilité d'écriture et flexibilité de la plateforme car le processeur est codé en VHDL et peut donc être modifié.

IV-1) Présentation de la plateforme

Le Plasma CPU est un domaine public qui a été créé en 2001 par Steve Rhoads. Ce microprocesseur RISC 32-bits, implémenté en VHDL tourne à 25 MHz sur une carte Xilinx Spartan-3E FPGA.

Le CPU est implémenté avec 2 ou 3 étages de pipeline, avec un étage supplémentaire pour la lecture et l'écriture mémoire.

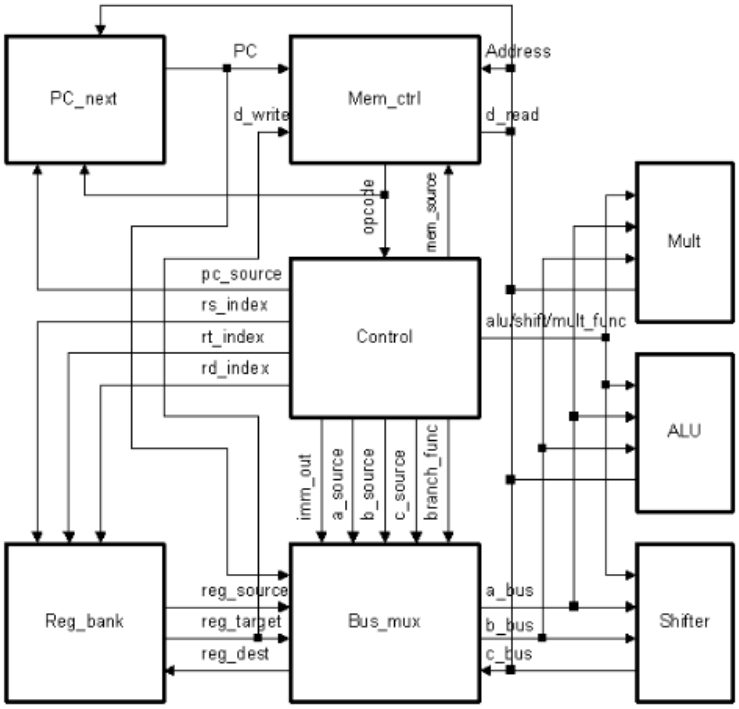

FIGURE 10 : DIAGRAMME BLOC DU PROCESSEUR PLASMA

Le processeur intègre 20 isa_custom disposant de deux entrées et une sortie permettant d'accélérer certains calculs plus rapides à réaliser en VHDL que par l'ALU et paramétrables par le programmeur et de 4 coprocesseurs permettant d'optimiser des gros calculs ou certaines tâches précises. Par exemple, le coprocesseur 4 est utilisé pour gérer l'affichage via la sortie VGA. Il utilise une

version modifiée du bloc VGA 640*480 de M. BORNAT. A chaque fin de calcul d'un pixel, celui-ci est envoyé au coprocesseur qui met à jour la RAM interne du VGA (cf figure 11).

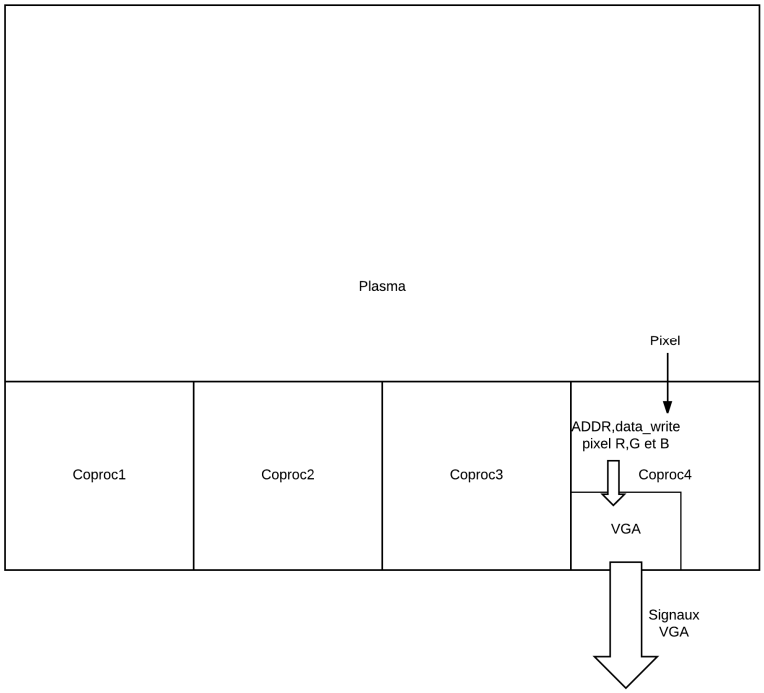

FIGURE 11 : SCHEMA BLOC INTERNE DU FONCTIONNEMENT DU VGA

IV-2) Optimisation du calcul via l'utilisation d'isa_custom

Comme expliqué précédemment, le calcul de la convergence d'un pixel mets en jeu trois multiplications sur des nombres de 32 bits pour vérifier que le module au carré du point est inférieur à 4 :

$$|(p = x + i * y)|^2 \leq 4 \Leftrightarrow x * x + y * y + 2 * x * y \leq 4$$

La première version écrite du plasma pour le calcul de la fractale utilisé la multiplication en C et donc un calcul relativement lourd au niveau du processeur. Cependant, en VHDL ce type d'opération peut s'effectuer en logique combinatoire. Il suffit d'envoyer à une isa_custom les deux données sur 32 bits, d'utiliser une variable temporaire sur 64 bits qui prend la multiplication des deux signaux d'entrée puis, de venir récupérer les 32 bit au bon endroit pour une valeur approché de la multiplication (cf Figure 12 ci-dessous).

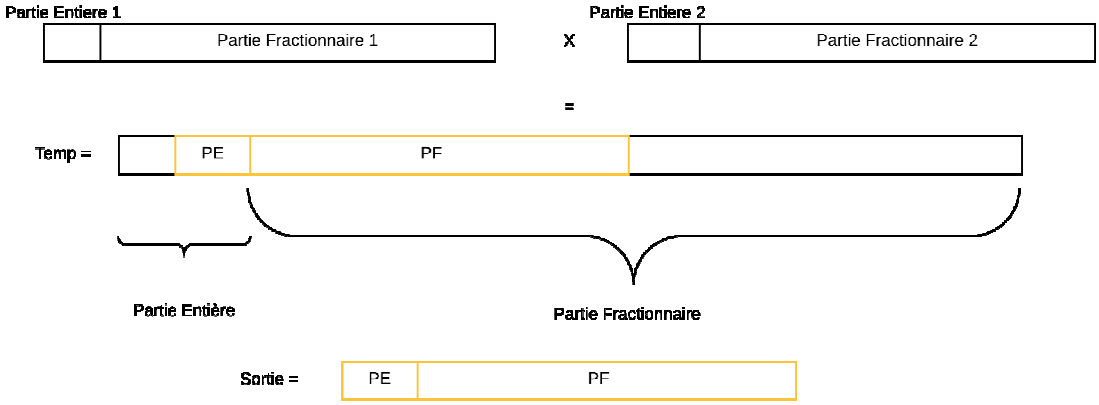

FIGURE 12 : SCHEMATISATION DU FONCTIONNEMENT D'UNE ISA_CUSTOM

Le gain de l'utilisation des isa_custom est tout simplement énorme. On passe de 26 min de rendu dans notre configuration choisie (4096 itérations) à seulement 1min10.

IV-4) Ajout d'une interface utilisateur

Une fois le programme C optimisé grâce à l'utilisation d'isa_custom, nous avons décidé d'implémenter une interface utilisateur via l'utilisation des 5 boutons poussoirs de la carte pour naviguer dans la fractale et pouvoir zoomer où l'on veut.

Pour cela, le processeur dispose d'un bus nommé GPIOA sur lequel on peut venir brancher les périphériques que l'on veut. Celui-ci est disponible à l'adresse 0x200000050. De ce fait, il nous suffit dans notre programme C de déclarer un pointeur sur cette adresse et de venir lire la donnée à cette valeur pour voir si les boutons sont appuyés ou non.

De plus, afin de ne détecter qu'un seul appui dans le programme C, nous avons rajouté un filtre de pulsation afin de ne pas détecter les rebonds du bouton poussoir et nous avons déclaré une fonction avec un booléen en static pour savoir si on a relâché où non le bouton à chaque appel de la fonction :

```
volatile bool buttonUp (volatile unsigned int *p ){
    static bool b=false;
    if(((*p) & MASK_UP) == MASK_UP){
        if (b==false)
        {
            b=true;
            return true;
        }
        else
            return false;
    }
    b=false;
    return false;
}
```

Ensuite, il nous a suffi de reprendre les calculs de zoom déjà effectués dans la partie VHDL pur et de les convertir en C.

IV-5) Parallélisation

Une fois le code C finalisé et optimisé, nous avons cherché à améliorer encore plus les performances en utilisant du parallélisme. L'objectif de cette partie était donc d'implémenter le maximum de CPU Plasma sur une seule carte FPGA avec chaque cœur travaillant sur une sous partie de l'écran de manière indépendante.

IV-5-a) Modification et optimisation du module VGA.

Comme expliqué précédemment, le VGA était initialement implémenté dans le coprocesseur4. Cependant, avec l'utilisation de plusieurs cœurs ce mode d'implémentation ne fonctionne plus car chaque cœur doit venir écrire dans le VGA au bon endroit de la RAM. L'idée a donc consisté à sortir le VGA du coprocesseur. Ainsi, le coprocesseur ne gère plus tous les signaux de sortie du VGA et se contente d'être une interface entre le bloc VGA et le processeur. Pour cela, il dispose de 4 signaux de sortie : OUTPU_1 qui est utilisé par le programme en C et donc inutile dans notre cas et de data_write qui est le signal d'activation de l'écriture sur le VGA, data_out la couleur à écrire et ADDR, l'adresse d'écriture.

D'autre part, afin de diminuer la taille occupée par le VGA sur la carte il a fallu procéder à quelques optimisations :

- Le VGA ne stocke plus la valeur du pixel mais le nombre d'itération effectué pour le calcul du pixel. De ce fait, il suffit d'utiliser un bloc de conversion en sortie du VGA pour transformer ce nombre en signal RGB. Dans notre phase de développement, nous avions fixé le nombre d'itération maximale à 256. Notre mémoire RAM dans le VGA est donc passée de 640*480*12bit/pixel à 640*480*7bits/itération.
- Nous avons aussi transformé la RAM pour qu'elle soit simple port avec priorité sur l'écriture. Grâce à cette modification, on prend beaucoup moins de place sur la carte car nous n'avons besoins que d'un seul bus de donnée. En revanche, cette modification fait apparaitre des artefacts à l'écran car il arrive que le VGA ait besoin d'une donnée dans la RAM pour l'affichage d'un pixel mais que celui-ci soit en cours d'écriture. Il apparait donc à l'écran, dans ce cas de figure, une couleur aléatoire.

IV-5-b) Modification du Plasma.

Comme expliqué dans le paragraphe précédent, nous avons modifié l'architecture du coprocesseur pour qu'il ne fournisse plus que le pixel, l'adresse et l'ordre d'écriture au VGA.

D'autre part, il a fallu rajouter un port generic de type string pour venir charger dans la mémoire boot le bon fichier code_bin.txt propre à chaque cœur.

Une fois cela fait, nous avons testé successivement avec deux cœurs, puis 4 cœurs et au final 8 cœurs. C'est à partir de 8 cœurs que nous sommes arrivés aux limites des ressources de la Nexys 4 dans notre configuration actuelle. En effet, nous avons dû désactiver deux cœurs pour réussir à faire passer notre design. Au final, la version avec 6 cœurs est fonctionnelle mais à un défaut d'affichages car elle était prévue pour 8 cœurs. Il y a donc une bande noire en bas de l'écran correspondant aux deux cœurs non implémentés. Par manque de temps, nous n'avons pas pu modifier notre design pour le faire tenir sur

6 cœurs avec un support complet de l'affichage. Nous avons donc effectués les tests de comparaison sur la version 100% fonctionnel à 4 cœurs (cf Figure 13).

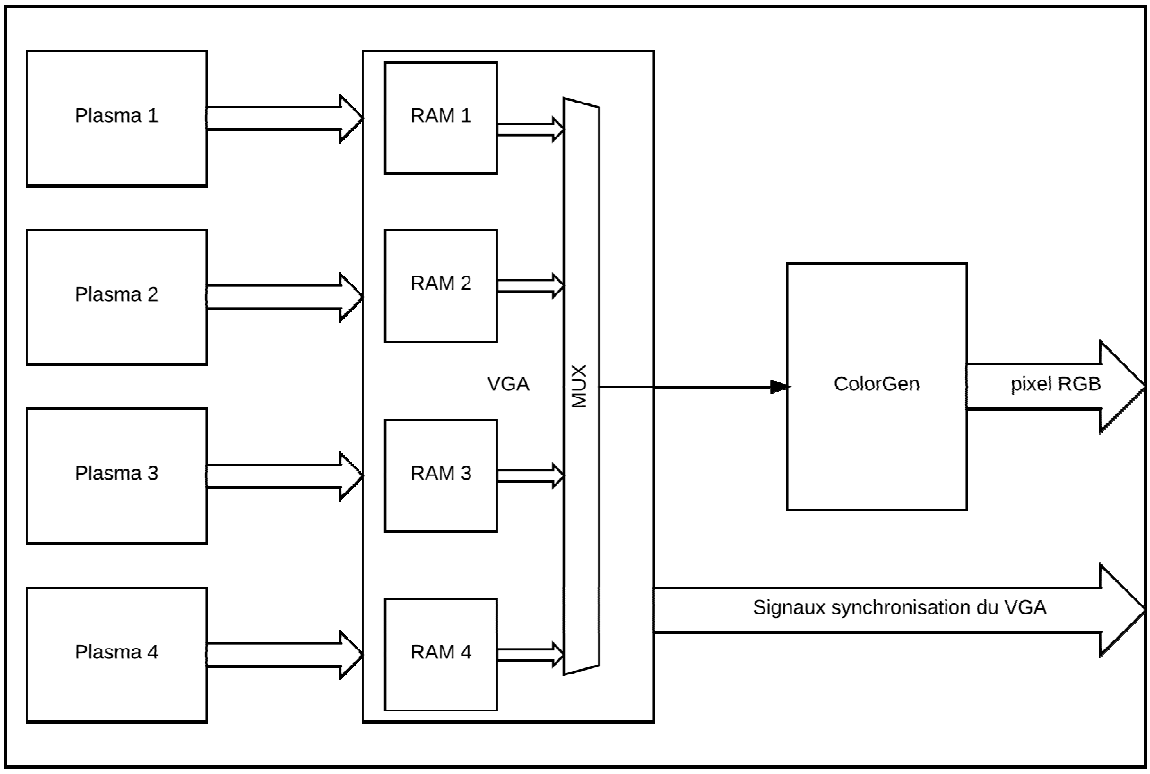

FIGURE 13 : SCHEMA BLOC DU PROCESSEUR MULTI CŒURS

Avec 4 cœurs, l'image est générée en 17 secondes ce qui correspond bien à un gain de 4 comparé à la version simple cœur.

On voit donc qu'il serait encore possible d'optimiser encore plus ce temps de calcul si on pouvait mettre plus de cœurs. Cependant, il faudrait trouver une nouvelle source d'optimisation au niveau du VGA tel que par exemple utiliser 8 cœurs mais simplement 4 RAM avec un multiplexeur, en entrée de ces RAM, qui sauvegarderai la valeur d'une des deux entré en cas d'écriture simultanée.

V-Bilan des améliorations et optique de developpement

V-1) Bilan des améliorations

Nous avons donc vu au cours de ce projet 3 plateformes différentes aux résultats et améliorations bien différentes. Celles-ci peuvent se résumer dans le tableau ci-dessous :

Plateforme	Logicielle (CPU)	VHDL pure	CPU plasma
Initialement	3 sec	3 sec	26 min puis 1 min 10 (optimisé
2 coeurs	2.5 sec	X	40 sec
4 coeurs	1.5 sec	X	17 sec
8 coeurs	<1 sec	X	X

On voit donc que la plateforme avec laquelle on a obtenu les meilleurs résultats est le VHDL pur. Au niveau logiciel, nous avons essayé de tenter de la parallélisation de l'algorithme mais cela n'a pas abouti.

Au niveau du Plasma, nous nous sommes rendu rapidement compte que cela n'offrait pas la puissance de calcul d'un vrai CPU ou du VHDL. En effet, le processeur est cadencé à 50MHz. On comprend qu'il ne fasse pas le poids contre un CPU actuel cadencé à plusieurs GHz. De plus, même si nous avons pu observer des gains significatif, en parallélisant et en optimisant, le développement en est d'autant plus long. En effet, il faudrait trouver d'autres façons d'optimiser le VGA et l'ensemble de l'architecture.

Au niveau du VHDL pur, les gains sont aussi efficaces mais le développement est d'autant plus long qu'il y a de modules de calcul. Une idée serait de généraliser la description et d'utiliser des programmes en C pour générer automatiquement les fichiers vhdl sans avoir à gérer pleins de copié/collés.

Par ailleurs, il y a d'autres plateformes à étudier que nous n'avons pas eu le temps d'aborder. Par exemple, le calcul sur GPU serait une piste pouvant donner de bons résultats. En effet, un GPU est composé de pleins d'unités de calcul indépendantes pouvant travailler en parallèle. On comprend tout de suite l'intérêt qu'aurait cette plateforme en associant un cœur de calcul à un tout petit ensemble de pixel.

VI-Conclusion

Pour conclure, nous pouvons dire que ce projet fut très intéressant. En effet, nous avons pu mettre en œuvre nos connaissances à travers l'étude d'un algorithme mais sur différentes plateformes. De plus, le fait que l'application soit visuelle a permis de rendre les phases de débogage plus simple mais aussi a rendu le développement plus ludique.

D'autre part, c'est la première fois que nous étions confrontés à un problème de ressources sur la carte FPGA Nexys 4. Il fut intéressant de chercher comment optimiser certains blocs comme par exemple le VGA afin de mettre plus de cœurs sur la carte.

Finalement, il aurait aussi été intéressant de travailler sur d'autres plateformes si nous avions eu plus de temps. Nous aurions aimez travailler par exemple sur l'implémentation sur GPU car c'est une plateforme complètement nouvelle pour nous.