Mehmed Blazevic

**Projet Mandelbrot**

**LPSC**

**Juin 2020**

Ici, vous pouvez trouver les différentes informations permettant de comprendre mon implémentation de Mandelbrot. Le schéma-bloc initial est le suivant :



On constate que chaque étape peut se faire de façon combinatoire. Les registres servant uniquement à garder l’état actuel du calcule. Deux blocs sont décrits en VHDL. Le niveau le plus bas est mandel\_iter. Celui-ci contient uniquement le calcule brut. Le niveau au-dessus est mandelbrot\_calculator. Celui-ci contient les éléments permettant d’entrer les données pour le calcul. Il contient également des éléments de contrôle comme une machine d’état.



Ici, on voit comment les signaux sont câblés. Ce schéma-bloc n’est pas détaillé, mais permet de comprendre comment fonctionne l’entité VHDL. La machine d’état permet de gérer les signaux de contrôles (start, ready et finish) et permet également la gestion des différentes itérations de calcul. Voici son fonctionnement :



Le schéma bloc ne le précise pas, mais c’est la machine d’état qui va présenter les données à calculer en entrée de mandel\_iter. Le calcule se fait sur 16 bits (initialement, puis 32) et j’utilise un DSP48 comme vu en cours pour les 2 équations qui comportent une multiplication.

Lorsque le fonctionnement de Mandelbrot a été validé au moyen de test-benchs basiques, j’ai commencé à faire le câblage général de mon système :

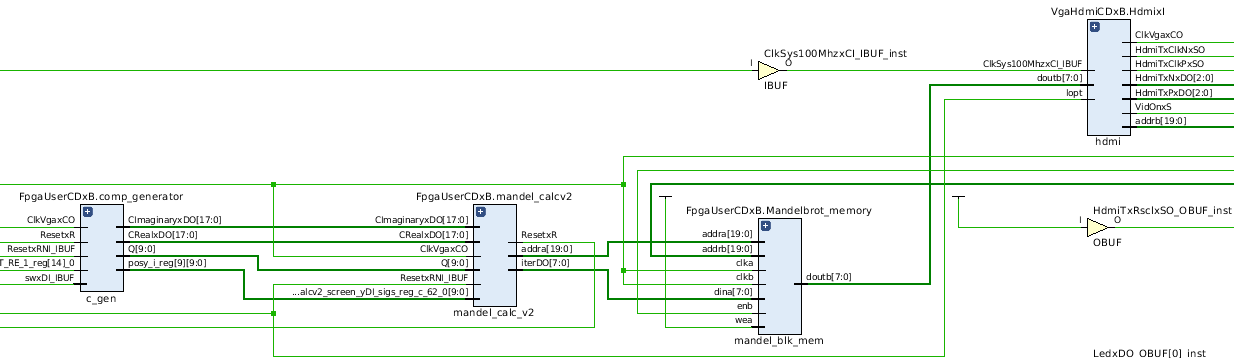


J’ai utilisé uniquement la clock VGA. swxDI[0] permet de faire un zoomOut et d’afficher l’image originel de Mandelbrot. Lorsqu’on appuie sur swxDI[1] (uniquement flanc montant), le signal zoomIn s’inverse et permet de lancer une animation de zoom. L’animation se fait au moyen d’un processus qui va simplement générer un signal à l’entrée zoomIn de c\_gen.

Le calculateur de Mandelbrot va demander des valeurs de nombres complexes à c\_gen avec la sortie MandelFinishDO à chaque fois que le calculateur a fini le calcul d’un pixel.

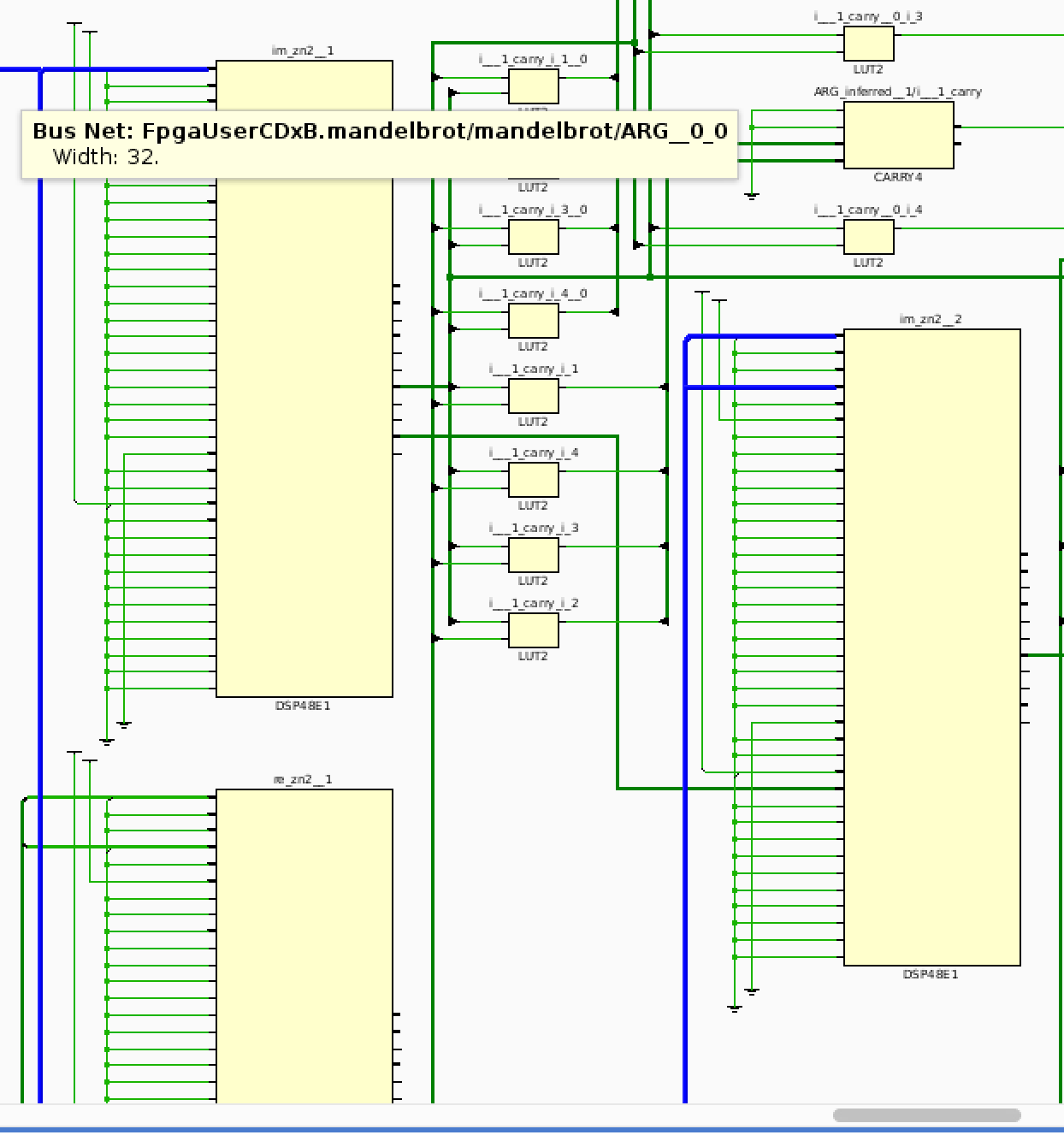
Un processus permet le comptage de l’adresse dans la mémoire à chaque fois que le calculateur fini un calcule. Le compteur compte jusqu’à 614400 (1024x600) et recommence. Le processus va également faire mapper le nombre d’itérations vers une valeur à stocker dans la mémoire. En réalité, c’est uniquement une adaptation de tailles qui est faites.

En sortie de mémoire, nous avons un simple mappage pour obtenir une valeur RGB et une conversion de Hcount et Vcount vers une adresse mémoire.

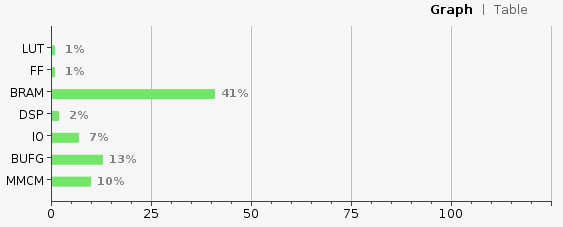


Sur ce schéma, on peut voir le câblage correspondant créé par Vivado. Ce n’est très lisible, mais le fichier original est dispo dans le repo. Une fois que le tout fonctionnait, j’ai décidé de passer mon système en 32 bits. Le 1er problème fut le fait que le DSP48 accepte un maximum de 18 bits de taille de donnée en entrée. J’ai donc enlevé les DSP initialement présent et écrit le calcul à la main.

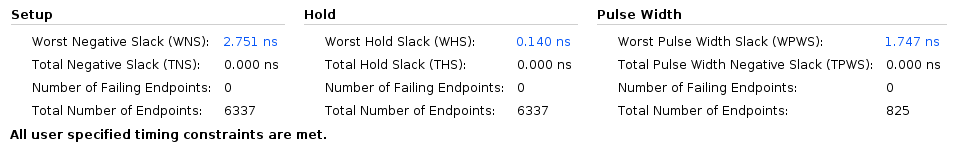
On peut constater que Vivado s’est débrouillé pour utiliser tout de même des DSP pour accélérer les calculs :



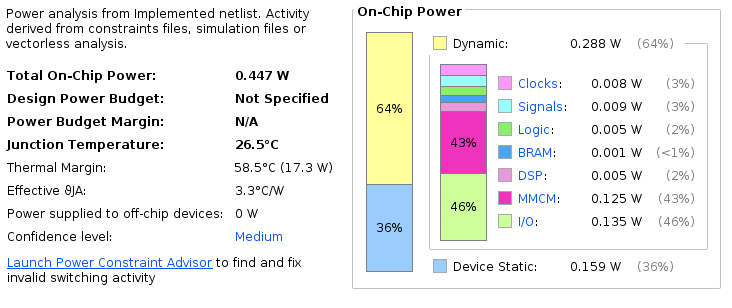
Voici l’utilisation de la FPGA :



Les timings :



Ainsi que la puissance :



Je n’étais pas très satisfait, car les calculs étaient assez lents et surtout non-déterministe à cause du nombre d’itérations. J’ai fait un système en pipeline qui a le même nombre de calculateurs que le nombre maximum d’itérations. Voici l’idée :



J’ai simplement chainé les circuits mandel\_iter et je les ai interconnectés avec des signaux. Pour commencer, les signaux de position d’écran, de fin et d’itérations ont été ajoutés.

La position de l’écran est donnée par c\_gen et est transmise de block en block, ainsi que les valeurs complexes initiales. Ainsi, chaque calcule est passé de block en block.

Les blocks vont également incrémenter la valeur d’itération lorsqu’ils font un calcule. Si un calcule est finit, la sortie finish du block actuel passe à 1. Ainsi, les bloque suivant, ne vont pas faire le calcul, mais vont simplement le passer au block suivant.

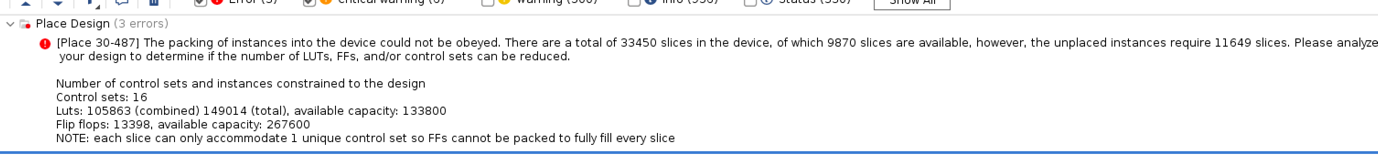
Au niveau du circuit mandelbrot, rien de particulier n’est fait à part le bon mappage entre les différents blocks et la mise à jour des entrées, en fonction des sorties précédantes.

On peut voir ce système comme un registre à décalage qui a chaque décalage effectue (ou non) un calcul, puis le transmet plus loin. L’avantage de ce système est qu’un pixel est calculé part flanc de clock, à l’exception des n premiers coups de clock.



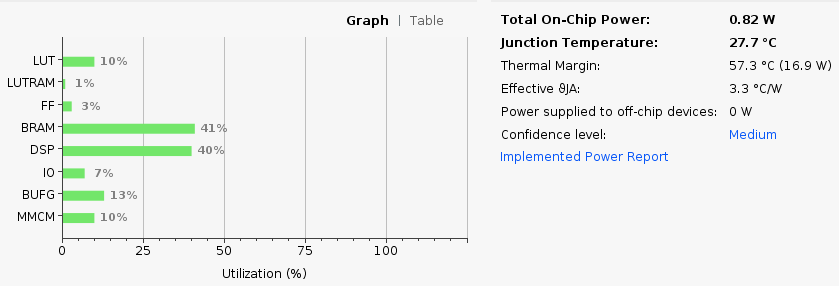
Voici le système complet dans sa seconde version. On constate que c\_gen fournit une valeur à chaque coup de clock. Mandelbrot possédant la position des pixels, il calcule directement l’adresse mémoire et il n’y a donc plus de processus permettant cela.

Je n’ai pas pu utiliser ce circuit en 32 bits notamment à cause de l’erreur suivante :

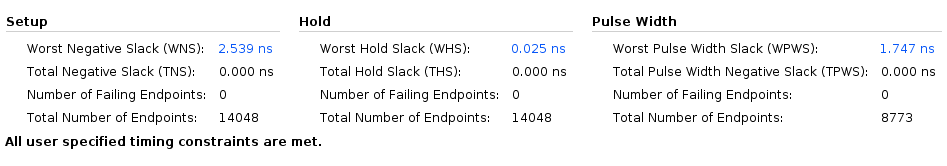


Pour remédier à cela, j’ai simplement baissé le nombre de bits de données. Actuellement le calcul se fait sur 18 bits (valeur maximum acceptable par un DSP48). J’ai donc également re-ajouté les DSP manuellement.

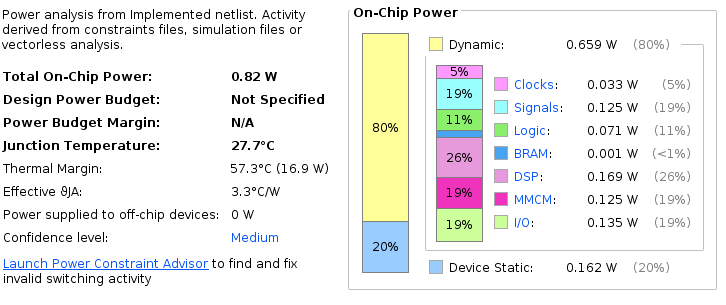
On constate que l’utilisation de de DSP a nettement augmenté par rapport au 1er circuit. Le nombre de LUT a également augmenté :



Voici les timings :



Et la puissance :



Ce rapport est probablement trop simple, mais j’espère qu’il est suffisamment clair. Quoi qu’il en soit, les 2 circuits fonctionnent et les fichiers permettant de programmer la FPGA sont présent dans le repo si vous voulez essayez ces circuits. Ce fut un plaisir de jouer avec cette FPGA et notamment d’améliorer le temps de traitement, sans pour autant augmenter la fréquence d’horloge.