Étude et réalisation d'une unité de calcul en virgule flottante pour un processeur destiné aux systèmes embarqués

Ing. C. FLÉMAL
ECAM – Bruxelles

Les FPUs (floating point unit) réalisent les opérations arithmétiques courantes de nombres en format virgule flottante. Ces nombres sont utilisés pour représenter des nombres réels et ont l'avantage de pouvoir couvrir une large gamme de valeurs. Cet article présente l'étude et la réalisation d'une FPU de base pour un processeur destiné aux systèmes embarqués. On y décrit la conception de ce circuit intégré. La FPU réalisée est validée et insérée dans un processeur de type RISC, architecture typique des systèmes embarqués. L'ensemble du travail fut réalisé à et pour l'unité DICE (Dispositifs Intégrés et Circuits Électroniques) de l'UCL.

Mots clés : nombre en format virgule flottante, unité de calcul flottant, processeur, systèmes embarqués

The FPUs (floating point unit) perform arithmetical operations on floating point format numbers. These numbers are used to represent real numbers and have the advantage of covering a wide range of values. This paper presents the design and construction of an FPU intended for a processor used in embedded systems. It describes the design of the integrated circuit. The FPU is implemented, validated and inserted into a RISC processor, a typical architecture of embedded systems. This work was done in and for the DICE unit (Devices Integrated and Electronics Circuits) at the UCL.

Keywords: floating-point number format, floating point unit, processor, embedded systems

1. Introduction

1.1 Contexte

Les systèmes embarqués

Les systèmes embarqués sont des systèmes composés d'un microprocesseur et qui doivent réaliser des tâches précises, comme par exemple du traitement du signal. Ils sont utilisés lors de multiples applications : téléphonie mobile, avionique, appareillage médical etc. Ils font partie intégrante des systèmes qu'ils contrôlent et donc sont souvent alimentés par batterie. Dès lors, leur consommation, comme celle de leur processeur, est critique. La vitesse de calcul de ce type de microprocesseur s'en trouve donc réduite. Certains systèmes embarqués effectuent des opérations de traitement du signal (transformée de Fourier, filtrage) comme par exemple pour la reconnaissance vocale, l'audio haute qualité et le contrôle de systèmes critiques (avion, fusée). Ce type d'opération peut requérir l'usage de nombres flottants.

Les nombres en format virgule flottante

Les nombres en format virgule flottante ont pour principal avantage une excellente dynamique : ils couvrent, pour un même nombre de bits, une étendue de nombres plus large que la représentation en nombres entiers. Ils sont donc utiles dès l'instant où l'information à traiter varie fortement vers des nombres extrêmement grands ou petits et sont généralement aussi utilisés pour représenter un nombre réel. Le format des nombres en virgule flottante est soumis à une norme, l'IEEE 754. La norme IEEE 754 spécifie :

- le format des nombres en virgule flottante : signe, mantisse et exposant ;
- la représentation des valeurs spéciales : infini, *not a number* et nombres dénormalisés. Les nombres dénormalisés sont des nombres qui sont trop petits pour être exprimés suivant le format classique ;
- les exceptions : *overflow*, *underflow*, inexact et opération invalide ;
- les modes d'arrondi : au plus près, vers zéro, vers $+\infty$ et vers $-\infty$;

Les FPUs

Une unité de calcul est l'un des constituants d'un microprocesseur. Les unités de calcul flottant (FPU) sont des unités arithmétiques qui effectuent des opérations sur les nombres flottants. Elles réalisent des opérations comme l'addition, la soustraction ou la multiplication. Elles se voient également confier des tâches de conversion vers et depuis d'autres formats.

Lorsqu'on réalise des calculs avec des nombres flottants, on a souvent besoin d'arrondir le résultat. Cela est dû au fait qu'on peut faire des opérations sur des nombres dont l'ordre de grandeur est fort différent. C'est pourquoi la norme IEEE 754 [6] a défini quatre manières d'arrondir, qui doivent donc être réalisées par les FPUs.

Les performances des FPUs sont définies d'une part, par leur fréquence de travail et, d'autre part, par leur latence et débit. La latence est le nombre de cycles nécessaires pour réaliser l'opération et le débit est le nombre de cycles qu'il faut attendre avant de demander une nouvelle opération à la FPU.

Pour une FPU destinée à un ordinateur de bureau où la rapidité est souvent prioritaire sur la consommation, la fréquence peut atteindre quelques GHz. Quand le but recherché est de limiter la consommation alors qu'on travaille en technologie CMOS, on va diminuer la fréquence afin de pouvoir réduire la tension d'alimentation. Comme la consommation évolue selon le carré de la tension, le gain de cette action est considérable. C'est pourquoi les FPUs destinées aux systèmes embarqués sont fréquemment limitées à des vitesses de fonctionnement moins élevées.

1.2 Objectifs

Notre but est la réalisation d'une unité de calcul flottant compatible avec la norme IEEE 754 32 bits, fonctionnant à 500 MHz et occupant une aire maximum de 20000 μm^2 (équivalent à 2 cm^2). Nous avons ensuite inséré l'unité de calcul dans un processeur de référence.

Tout au long de ce travail, la technologie CMOS (Complementary Metal Oxide Semiconductor) 65 nm low power est utilisée. Dans cette technologie,

l'élément le plus limitant en fréquence est la RAM (mémoire vive). Sa fréquence caractéristique est de 500 MHz. Il n'est dès lors pas nécessaire de fonctionner à plus haute fréquence. C'est pourquoi la fréquence de fonctionnement de la FPU réalisée est de 500 MHz. De plus, dans cette technologie, le processeur complet devrait occuper une aire de l'ordre de 40000 μm^2 dont la FPU peut représenter la moitié de l'aire, d'où l'objectif des 20000 μm^2.

La FPU réalisée effectue les opérations d'addition, soustraction et multiplication. Les opérations de division et racine carrée sont nettement plus complexes et deux solutions étaient envisageables : soit on réalisait une implémentation simple mais peu performante, soit on étudiait la façon de réaliser ces opérations de manière performante mais sans les implémenter. Le choix a été porté sur la deuxième solution et une recherche sur le sujet a été réalisée. Kwon Taek-Jun et Draper Jeffrey [7] proposent une architecture combinée de multiplication, division et racine carrée basée sur une implémentation de la division proposée par Albert A. Liddicoat et Michael J. Flynn [8]. Elle pourrait servir de réflexion de base pour la mise en œuvre d'une telle structure.

En résumé, le but n'est pas de réaliser une FPU haute performance travaillant à haute fréquence (quelques GHz) afin d'offrir un débit important au détriment de la consommation, mais bien de réaliser une FPU qui ne présente ni une aire ni une consommation importantes à la fréquence de 500 MHz, tout en offrant le maximum de ses possibilités de calcul.

1.3 Flot de conception

Lorsqu'on réalise un circuit électronique digital, il s'agit typiquement de réaliser plusieurs étapes.

La description théorique du circuit

Il convient d'abord de spécifier de manière claire l'algorithme que l'on veut réaliser. Ensuite, il faut choisir les composants de base (additionneur, registre, etc) que l'on va employer pour réaliser cet algorithme. Finalement, l'étape d'ordonnancement permet de décider de la gestion de ces composants à chaque cycle d'exécution. En effet, une même ressource peut être

utilisée plusieurs fois pour réaliser l'algorithme mais à des cycles différents (*Hardware multiplexing*).

La description hardware du circuit

Des langages spécialisés, les HDL (*Hardware Description Language*), permettent de décrire le circuit logique que l'on veut implémenter sur base de l'étude faite précédemment.

La validation fonctionnelle

Grâce à la simulation du circuit décrit en HDL, il est possible de vérifier le bon fonctionnement du circuit. Cette simulation consiste à mettre des valeurs en entrée du circuit, et à vérifier que la sortie de ce même circuit est conforme aux attentes. Il faut donc préalablement définir les couples entrées-sorties qui seront utilisés pour faire les tests. Ces couples sont appelés vecteurs de tests.

La synthèse

La synthèse consiste, sur base de la description du circuit logique, à choisir les éléments logiques simples (portes logiques, multiplexeurs, etc) qui vont constituer le circuit. L'outil de synthèse va optimiser le choix de ces éléments afin de satisfaire les contraintes temporelles et en surface qu'on lui soumet. En effet, il dispose de librairies de composants qui ont des caractéristiques précises : un petit composant occupera moins d'aire mais aura une réponse plus lente ; au contraire, un composant plus grand sera plus rapide. L'outil va donc choisir parmi tous ses composants ceux qui permettent de réaliser le circuit pour une surface et une fréquence qu'on lui impose. S'il parvient à réaliser le circuit, il crée une *Netlist*, reprenant l'ensemble des éléments et la façon dont ils se lient.

Le placement-routage

Sur base de la *Netlist*, l'outil de placement-routage détermine la position des éléments logiques sur le substrat de silicium et les relie correctement entre eux à l'aide de pistes, c'est l'étape de routage.

La vérification fonctionnelle

Il convient enfin de vérifier que la simulation fonctionnelle réalisée précédemment donne toujours les résultats escomptés après les étapes de synthèse et placement-routage qui modélisent le fonctionnement physique du circuit.

2. Description théorique du circuit

2.1 Fonctionnement d'une FPU

Lors d'un calcul en nombre flottant, trois étapes doivent être réalisées :

-la pré-normalisation prépare l'opérande à subir l'opération arithmétique. Elle sert à extraire les exposants et mantisses de manière à pouvoir ensuite réaliser l'opération comme une opération entière. Cette étape est différente selon le type d'opération arithmétique réalisée.

-l'opération arithmétique proprement dite dont le rôle est le calcul de la mantisse finale non normalisée et la détermination du bit de signe.

-la post-normalisation gère le mode d'arrondi, les exceptions et le formatage en nombre flottant. Le rôle de la post-normalisation est de récupérer le résultat de l'opération : le signe, la mantisse et l'exposant et de représenter la réponse sous la forme d'un nombre flottant conforme à la norme IEEE. Cette étape diffère légèrement d'une opération à l'autre. C'est l'étape la plus complexe du point de vue algorithmique.

2.2 Analyse des algorithmes

Pour chaque étape (pré-normalisation, opération, post-normalisation), un algorithme a été défini. Il faut maintenant optimiser cet algorithme afin qu'il ait les meilleures performances possibles dans le cas de notre application particulière. Sur base des algorithmes, le besoin en ressources de chacun d'eux a été déterminé.

L'étude du besoin en ressources permet d'analyser les différentes possibilités d'implémentation. En effet, certaines ressources sont utilisées plusieurs fois lors d'une opération. Cela offre deux possibilités : soit on réutilise

plusieurs fois une même ressource lors de cycles différents (gain en surface), soit on implémente autant de ressources qu'il y a d'utilisation de celle-ci (gain en temps).
Le fait de décider cycle par cycle de l'utilisation des composants s'appelle l'ordonnancement. Autrement dit, l'ordonnancement permet d'attribuer, à chaque cycle d'horloge, les registres et les ressources de calculs nécessaires à la bonne réalisation de la tâche demandée.

Pour pouvoir correctement évaluer cet ordonnancement, il y a également lieu de connaître les caractéristiques temporelles des ressources utilisées afin de pouvoir estimer l'importance du chemin critique du circuit qui permettra de déterminer le nombre de cycles d'horloge nécessaire à la réalisation de l'algorithme. Pour estimer la valeur d'un chemin, il faut déterminer le nombre maximum de portes logiques en série sur ce chemin; c'est ce qu'on appelle sa profondeur. Le tableau 1 fournit les valeurs des profondeurs des ressources utilisées dans ces algorithmes (*n* étant le nombre de bits sur lequel elles travaillent et *i* le nombre d'entrée) [3].

	Taille	Profondeur
Additionneur générique	n	n
Multiplexeur	n	log(i)
Shifter	n log(n)	log(n)
Leading_zero	n	log(n)
Comparateur	n	log(n)

Tableau 1: Taille et profondeur des ressources utilisées

Dans les systèmes 32 bits, l'opération typique la plus complexe que l'on réalise en un seul cycle est un additionneur 32 bits, ce qui correspond à une profondeur d'environ 32 portes. Cette profondeur de 32 portes servira donc de référence pour l'ensemble de la FPU. Pour pouvoir être réalisée en un seul cycle, une opération ne pourra donc pas dépasser une profondeur de 32 portes.

Il ne faut cependant pas oublier que l'outil de synthèse va réaliser une optimisation qui tiendra compte des contraintes qu'on lui impose. En effet, l'outil peut choisir des cellules et des portes logiques qui permettent d'atteindre les contraintes en surface et/ou en fréquence. Différentes cellules qui réalisent les mêmes fonctions logiques peuvent avoir des tailles différentes : soit parce qu'elles font leurs calculs de manières différentes, soit

parce qu’elles ont des vitesses de calcul différentes. Or, plus les contraintes en fréquences sont strictes, plus le circuit doit aller vite. L’outil de synthèse va donc choisir des cellules plus rapides (et plus grandes) ou des cellules qui réalisent leur calcul d’une manière plus sophistiquée (et occupant plus d’aire). Dès lors, si on impose des fréquences de plus en plus élevées, la surface ira également en croissant, impactant également la consommation. C’est pour cette raison qu’en première estimation, on définit la profondeur du chemin critique en nombre de portes. Ce n’est qu’après synthèse qu’on connaîtra réellement les paramètres de surface et de vitesse.

Exemple 1 : Opération arithmétique addition-soustraction

La figure 1 présente l’algorithme pour l’opération arithmétique d’addition-soustraction (entre la pré-normalisation et la post-normalisation). Cet algorithme sera expliqué sans entrer dans tous les détails. Les entrées *fracta* et *fractb*, issues de la pré-normalisation, sont des nombres positifs sur 28 bits dont les signes associés sont *signa* et *signb*. La sortie *fract* et le signe associé sont destinés à la post-normalisation.

Etape1a : Sélectionner addition ou soustraction et déterminer le plus petit opérande

Etape1b : Aiguiller soit vers additionneur soit vers soustracteur

Etape2 : Déterminer le signe et réaliser l’opération

Etape3 : Sélectionner le résultat

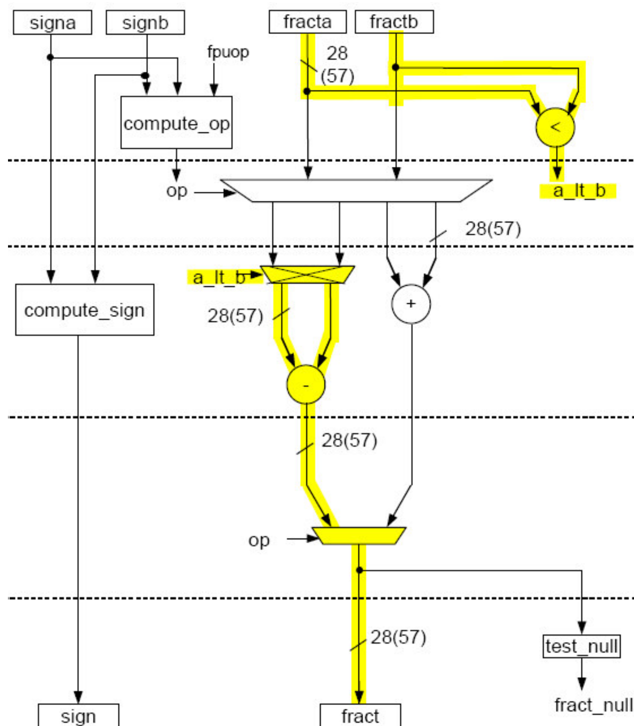

Figure 1: Algorithme d'addition-soustraction

Comme cet opérateur peut effectuer deux opérations différentes (addition ou soustraction), une première étape (étape 1a) consiste à savoir quelle opération sera réalisée en fonction des signes et de l'opération demandée. Par exemple, si une soustraction est demandée et que le deuxième nombre est négatif alors on effectuera une addition. Le comparateur et le module compute_op déterminent l'opération à effectuer et l'ordre des opérandes.

Le chemin critique, c'est-à-dire celui nécessitant le plus de temps, apparaît en surligné sur la figure 1. Il est constitué d'une comparaison de 28 bits (cinq portes logiques) suivie de l'additionneur 28 bits (28 portes logiques), soit un total de 33 portes logiques, ce qui dépasse la limite de ce qui est réalisable en un seul cycle. La synthèse autorisant une certaine optimisation, cette opération sera donc réalisée sur un cycle. Notons que l'étude a aussi été faite pour les nombres 64 bits, dans ce cas les opérandes d'entrées sont sur 57 bits.

Exemple 2 : Post-normalisation

Le schéma de la post-normalisation est présenté à la figure 2, mais nous ne l'analyserons pas dans tous les détails.
Le chemin critique de cet algorithme est de 53 portes. Cette valeur est supérieure aux 32 portes de références.

Deux solutions s'offrent à nous :

- réaliser la post-normalisation en deux cycles. Cela permettra de partager des ressources d'un cycle à l'autre et de moins contraindre les ressources en fréquence, évitant ainsi une augmentation de leur surface. Mais cela demandera l'ajout d'une barrière de registre dans le module de post-normalisation, ce qui va consommer de la surface.
- réaliser la post-normalisation en un seul cycle. Cela permettra d'optimiser le temps de calcul de la FPU mais cela induire une forte contrainte en fréquence car la limite des 32 portes est dépassée. La surface de la post-normalisation va donc fortement augmenter.

Le choix a été porté sur la deuxième solution, car elle permet d'effectuer les opérations en moins de cycles. Une analyse du comportement en fréquence du module a permis d'estimer la surface utilisée par celui-ci. À 500 MHz, le module occupe une surface de 4550 µm². Cette taille ne représente que le cinquième de la taille totale permise pour l'ensemble de la FPU et il ne

semble donc pas opportun de dégrader ses performances en passant à deux cycles pour la post-normalisation.

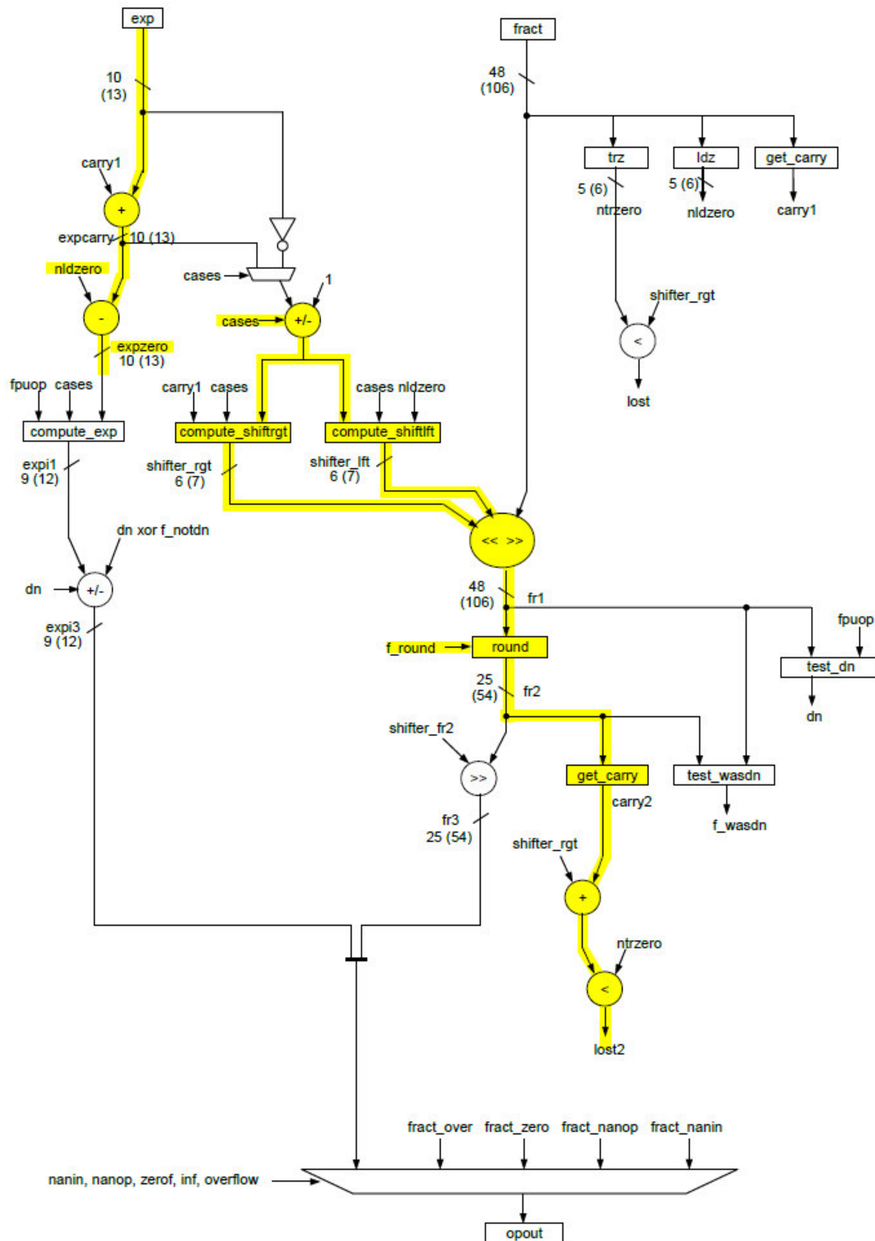

Figure 2: Algorithme de la post-normalisation

2.3 Résultats de l'analyse

Le nombre de cycles nécessaires pour chaque algorithme est présenté dans le tableau suivant.

	Nombre de cycles en 32 bits
Prenorm Add-sub et Mul	1
Add	1
Mul	2
Postnorm	1
Total pour Add-sub	3
Total pour Mul	4

Tableau 2: Nombre de cycles pour chaque algorithme

3. Description hardware du circuit

L'entièreté de la FPU est codée en Verilog et simulée à l'aide du logiciel Modelsim. Les trois étages de la FPU sont codés et testés séparément, ensuite ils sont mis en commun afin de réaliser la FPU complète.

Tous les modules (pré et postnormalisation, opérations) ont été codés en Verilog et le fonctionnement de chacun d'eux a été testé. Cela a permis une première détection d'erreurs de codage. Une fois validés, les différents modules ont été associés à l'aide du module FPU et doivent à présent être validés en profondeur. Cette étape est très importante car la FPU doit être opérationnelle et 100 % compatible avec la norme IEEE 754. Il faut donc à la fois vérifier les résultats fournis par les différentes opérations mais également les bits de flags de sortie qui expriment les exceptions. Le paragraphe suivant est entièrement consacré à cette validation.

4. Validation fonctionnelle

Un environnement de test a été créé et la FPU est validée par le passage de vecteurs de test. Ces vecteurs sont composés de nombres flottants à mettre en entrée de la FPU et des résultats attendus en sortie de la FPU. Il s'agit

donc de valeurs pré-calculées. Ces vecteurs sont générés par la librairie Testfloat [5] réalisée par Softfloat et sont compatibles avec la norme IEEE.

Le module de test est composé, d'une part d'une mémoire contenant les instructions, les opérandes d'entrées et les résultats attendus et d'autre part d'un *pipeline* qui simule le fonctionnement de la FPU afin de comparer les résultats obtenus et attendus au bon moment.

La vérification consiste d'une part, à s'assurer que l'opérande de sortie ainsi que tous les *flags* correspondants aux exceptions et à certains cas rares sont exacts, c'est-à-dire : inexact, overflow et underflow, infini, zéro, qNaN et sNaN.

La librairie Testfloat ne permet pas de générer directement des vecteurs de test. Elle permet seulement de calculer les résultats de nombreuses opérations en virgule flottante et ce de manière compatible avec la norme IEEE 754. C'est à l'utilisateur de choisir les opérandes d'entrées. Le choix de ces opérandes est primordial : il faut à la fois tester de nombreux cas communs mais également tous les cas critiques comme les nombres dénormalisés, les infinis et les *NaN*s. C'est pourquoi plusieurs fichiers de test seront réalisés : certains pour les cas communs, d'autres pour les cas critiques.

Trois tests ont été réalisés : test d'addition/soustraction, test de multiplication, test d'addition/soustraction et multiplication.

Les deux premiers tests permettent de vérifier la fonctionnalité algorithmique de la FPU. Le test d'addition/soustraction comprend 4800000 vecteurs de test, celui de la multiplication 2400000. Pour ces deux tests, un tiers des vecteurs réalisent des opérations de cas critiques et les deux tiers restants des cas généraux. Le dernier test permet de détecter d'éventuels dysfonctionnements dans le pipeline, 500000 vecteurs de test y sont consacrés. Tous ces vecteurs sont correctement passés, c'est-à-dire que les résultats fournis par la FPU correspondent à ceux attendus. La conception de la FPU peut donc être validée et on peut également garantir qu'elle respecte bien la norme IEEE 754.

5. Synthèse

La synthèse permet de générer une *Netlist*. Celle-ci détermine l'ensemble des circuits logiques de base qui sont nécessaires pour réaliser le circuit et explicite la façon dont ils se connectent. Cette *Netlist* est générée à partir de la description HDL du circuit, écrite ici en Verilog. La FPU a été synthétisée à l'aide du logiciel Design Compiler [12].

Le tableau 3 présente la surface totale de la FPU. On observe que cette surface se situe sous les 20 000 μm2 : cet objectif est donc atteint. La FPU a été synthétisée autour de la fréquence de fonctionnement (500 MHz). De 400 à 500 MHz, la surface augmente de 0,2 %. De 500 à 600, elle augmente de 2,4 %. On peut donc en conclure que la FPU ne se trouve pas à une fréquence de fonctionnement critique. Le chemin critique de la FPU se trouve comme prévu dans le module de post-normalisation, ce qui montre à nouveau qu'il faudrait se pencher sur ce module si l'on souhaitait améliorer la FPU existante.

Fréquence [MHz]	Surface [μm^2]
400	16243
500	16274
600	16664

Tableau 3: Résultat de la synthèse

6. Placement routage

6.1 Insertion de la FPU dans le processeur de référence

Avant de réaliser l'étape de placement-routage, la FPU est insérée dans un processeur scalaire RISC (*Reduced Instruction Set Computer*) à cinq étages, d'une structure similaire au MIPS [9] [10] (*Microprocessor without Interlocked Pipeline Stages*). Cette architecture, pipelinée, est souvent utilisée dans les systèmes embarqués. Ce processeur est fourni par le laboratoire DICE de l'UCL ; l'objectif est d'y insérer la FPU. Cela demande trois étapes principales :
- l'extension du jeu d'instructions : il est nécessaire de créer de nouvelles instructions pour réaliser les opérations sur la FPU.
- la modification du pipeline ; la FPU étant pipelinée, il faudra en tenir compte et modifier l'architecture du processeur en conséquence.

- l'insertion de la FPU proprement dite : des signaux de contrôle devront permettre de faire travailler la FPU au bon moment et de récupérer son résultat.

Ces étapes sont ensuite testées pour vérifier que l'insertion s'est bien déroulée. La figure 3 présente le résultat d'un test d'addition :

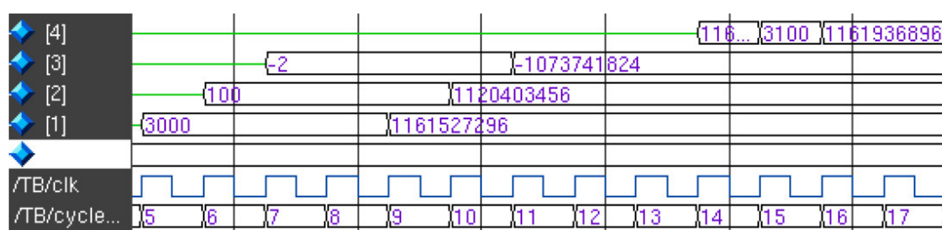

Figure 3: Diagramme temporel issu de la simulation

Les nombres en mémoire sont : 3000 en mem(0), 100 en mem(1) et -2 en mem(2). Le programme commence par chercher en mémoire les trois nombres et les convertit en format flottant. Ensuite, il additionne les nombres contenus dans r1 (3000) et r2 (100) et met le résultat en r4 (3100). Ce résultat est ensuite converti en entier ce qui permet de vérifier plus aisément qu'il est correct.

6.2 Réalisation du placement-routage

Comme la FPU a été vérifiée et que son insertion dans le processeur est fonctionnelle, le placement-routage de l'ensemble peut être effectué. Cette étape consiste, à partir de la Netlist issue de la synthèse, à placer les différents composants sur le substrat en silicium (placement) et à les relier entre eux (routage). Cette étape prend en compte les délais engendrés par les pistes et les composants, elle permet donc de vérifier si le circuit physique est réalisable. Le placement-routage a été réalisé à l'aide du logiciel Cadence. Il fourni en sortie une image du circuit (*layout*) et un fichier de description en Verilog. Ce fichier permet de réaliser une simulation du circuit placé et routé afin de vérifier qu'il est toujours fonctionnel.

6.3 La représentation du layout

La figure 4 représente les sept étages de routage placés au-dessus des composants de bases qui constituent le processeur.

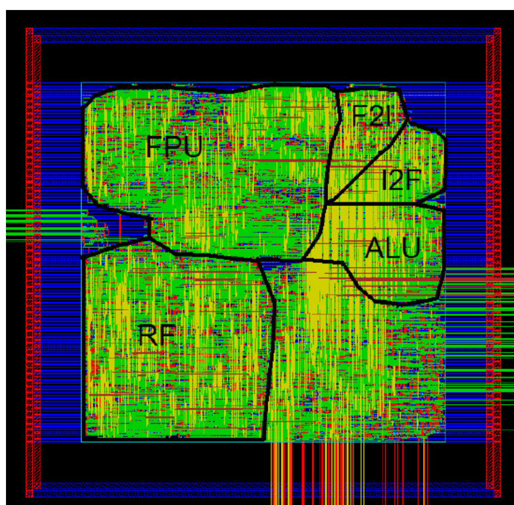

Figure 4: Layout du circuit

L'ensemble occupe une aire de 55000 μm2. La FPU se trouve dans le coin supérieur gauche, à coté des deux modules de conversion (*F2I* et *I2F*). Le reste constitue le processeur de base, avec son ALU (unité arithmétique et logique) et son banc de registre (*RF*). On peut remarquer qu'un banc de 32 registres de 32 bits occupe déjà, dans notre cas, un quart de la surface occupée par le processeur.

7. Vérification fonctionnelle

Un test du fonctionnement vérifiant toutes les opérations a été mené. Lors de ce test, on voit apparaître des délais et oscillations, conséquences de la modélisation des caractéristiques physiques du circuit. Ce test constitue la dernière étape de validation du circuit réalisé.

La figure 5 représente l'addition du contenu du registre 2 (100) avec lui-même. Le résultat (200) est replacé dans ce registre. Le nombre 100 est d'abord converti en nombre flottant et passe dans la FPU. Ensuite, il est converti en nombre entier et est mis dans le registre 2.

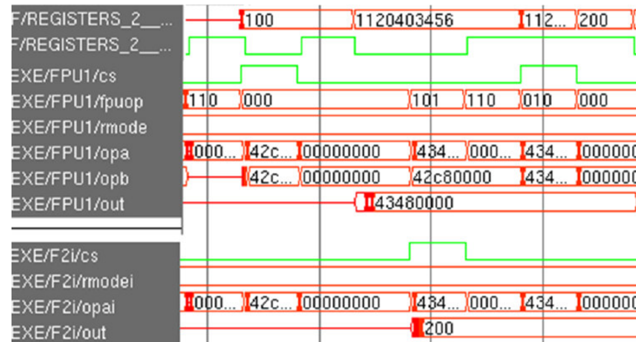

Figure 5: Simulation après placement-routage

8. Comparaison avec d’autres FPUs

8.1 Comparaison avec des FPUs d’ Opencores

Le site d’Opencores propose de partager des descriptions HDL de composants multiples. On y retrouve les FPUs [14] [13] [2] [1]. Deux FPUs ont été choisies car elles étaient les plus semblables à la FPU réalisée. Ces FPUs ont été décrites sans tenir compte des aspects de minimisation d’aire et de consommation. Lorsqu’on les synthétise à la fréquence de 500 MHz et en technologie CMOS 65 nm low power, on peut constater qu’elles occupent 25 % d’aire de plus que la FPU réalisée. L’analyse des algorithmes et le travail sur ceux-ci sont donc concluants.

Design	Aire (μm^2)	Latence/débit	
		a + b	a x b
FPU Al-Eryani	23144	7/7	12/12
FPU Usselman	21566	3/1	3/1
FPU réalisée	16274	3/1	4/1

Tableau 4: Comparaison avec des FPUs d'Opencores

8.2 Comparaison de FPUs de processeurs

Deux FPU de processeurs courants sont présentés ici. Leurs latences sont identiques voire légèrement supérieures à celles de la FPU réalisée. Les améliorations apportées à ces processeurs se situent principalement au niveau de la fréquence de travail, améliorations rendues possibles par une

implémentation optimisée au niveau 'transistor' mais également par l'utilisation d'une technologie différente. En effet, comme dit plus avant, la FPU réalisée fait appel à une technologie *low power 65 nm* qui permet de réduire la consommation tandis que certains processeurs (le Intel Core i7) travaillent en technologie 45 nm ce qui leur permet tout naturellement d'atteindre de plus hautes fréquences.

La conclusion de cette comparaison est donc qu'au point de vue latence et débit, la FPU réalisée correspond bien à ce qui se fait dans les autres FPUs. De plus elle respecte l'exigence de surface imposée. Elle pourrait travailler à plus haute fréquence mais ce n'est pas le but recherché : l'objectif est bien d'avoir une FPU fonctionnant à 500 MHz sans avoir une surface ou consommation importantes.

Design	Fréquence (GHz)	Latence/débit	
		a + b	a x b
Intel Core i7	3.2	3/1	4/1
IBM Power6	4.7	6/1	6/1
FPU réalisée	0,5	3/1	4/1

Tableau 5: Comparaison de FPUs de processeurs

9. Conclusion

La FPU réalisée est maintenant utilisable dans un processeur. Ses caractéristiques (fréquence de fonctionnement, surface et consommation) sont adaptées à un processeur destiné aux systèmes embarqués. Elle supporte la norme IEEE 754, ce qui la rend facilement compatible avec l'environnement dans lequel elle va s'intégrer.

Elle peut également être améliorée : l'ajout des opérations de division et racine carrée avec partage des ressources du multiplieur, l'ajout des opérations de conversions ou encore une compatibilité 64 bits permettant de gérer à la fois les formats 32 et 64 bits de la norme IEEE.

Enfin, l'étude algorithmique qui a été menée est réutilisable dans d'autres contextes, comme par exemple pour l'insertion d'une FPU dans un FPGA (field-programmable gate array).

10. Sources

[1] AL ERYANI Jidan. *Floating point unit*, 2006.

[2] AL ERYANI Jidan. *FPU::Overview*. Opencores. http://www.opencores.org/project,fpu100, 2007

[3] BERG Christoph, JACOBI Christian & KROENING Daniel. *Formal verification of a basic circuits library*. In In Proc. of IASTED Int. Conf. on Applied Informatics, Innsbruck (AI 2001). ACTA Press, 2001.

[4] CATOVIC Edvin. *Grfpu-high performance ieee-754 floating-point unit*. 2004.

[5] HAUSER John., *Testfloat*. http:*//www.jhauser.us/arithmetic/TestFloat.html*, 2010

[6] IEEE, *Ieee standard for floating-point arithmetic*. IEEE Std 754-2008, pages 29–58, 2008.

[7] KWON Taek-Jun & DRAPER Jeffrey. *Floating-point division and square root using a taylor-series expansion algorithm*. Microelectron. J., 40(11):1601–1605, 2009.

[8] LIDDICOAT Albert A. & FLYNN Michael J. *High-performance floating point divide*. In In Proceedings of the Euromicro Symposium on Digital System Design, pages 354–361, 2001.

[9] MIPS TECHNOLOGIES, Inc. *MIPS32TM Architecture For Programmers Volume I : Introduction to the MIPS32TM*, March 2001. Revision 0.95.

[10] MIPS TECHNOLOGIES, Inc. *MIPS32TM Architecture For Programmers Volume II : The MIPS32TM Instruction Set*, June 2003. Revision 2.00.

[11] SODERQUIST Peter & LEESER Miriam. Floating-point division and square root: Choosing the right implementation, 1996.

[12] SYNOPSYS, Design compiler. http://www.synopsys.com/tools/implementation/rtlsynthesis/pages/designcompiler2010-ds.aspx, 2010

[13] USSELMAN Rudolf. *Open Floating point unit*. The Free IP Cores Projects : www.opencores.org, September 2000.

[14] USSELMAN Rudolf . *Floating Point Unit::Overview*. Opencores. http://www.opencores.org/project,fpu. 2009