Implantation en virgule fixe d'un opérateur de calcul d'inverse à base de Newton-Raphson, sans normalisation et sans bloc mémoire

Erwan LIBESSART, Matthieu ARZEL, Cyril LAHUEC, Francesco ANDRIULLI

IMT Atlantique
Technopôle Brest-Iroise CS 83818, 29238 Brest Cedex 3, France
prénom.nom@imt-atlantique.fr

Résumé – La division est une opération courante dans le domaine du traitement du signal. Dans le cas de l'implantation matérielle, il est préférable de calculer dans un premier temps l'inverse du diviseur, puis de multiplier ce résultat au dividende. L'algorithme de Newton-Raphson peut être utilisé pour calculer cet inverse, que ce soit en représentation flottante ou fixe. Ce papier propose une implantation en virgule fixe d'un opérateur de calcul d'inverse utilisant l'algorithme de Newton-Raphson. Cette implantation se démarque par l'absence de blocs mémoire et d'une normalisation des entrées et cherche à maximiser le débit sur FPGA.

Abstract – Division is a common operation in signal processing. For its hardware implementation, the divisor's reciprocal is first calculated and then multiplied by the dividend. The Newton-Raphson algorithm can be used to compute the reciprocal, either in floating or in fixed-point format. This paper presents a Newton-Raphson implementation for a fixed-point reciprocal operator. The scaling-less and ROM-less implementation maximizes the throughput on FPGA target.

1 Introduction

L'usage de la division est souvent un passage obligé dans les algorithmes de traitement du signal mais dont l'implantation matérielle cause souvent des problèmes de ressources. La stratégie habituelle consiste en deux étapes : calculer l'inverse du diviseur puis multiplier ce résultat par le dividende. Ce calcul d'inverse peut se faire en représentation flottante ou fixe. Pour la première possibilité, la plus grande difficulté revient à inverser la mantisse, qui est comprise, par définition, dans l'intervalle $[1, 2[$. Des méthodes d'évaluation reposant sur des tables ou des méthodes itératives comme l'algorithme SRT peuvent être utilisées, mais les ressources et latence nécessaires deviennent conséquentes quand la précision requise augmente [1]. Une autre solution possible est d'utiliser l'algorithme de Newton-Raphson, qui est une méthode itérative où la précision est environ doublée à chaque itération. Toutes ces méthodes peuvent être employées dans le cas d'une représentation en virgule fixe. Il suffit de normaliser les données afin de se ramener au cas flottant. Dans le cas de la méthode Newton-Raphson, la normalisation peut se faire vers les intervalles $[1, 2[$ ou $[0.5, 1[$ et la dénormalisation est donc nécessaire en sortie de l'opérateur [2, 3]. La première approximation de cet algorithme se calcule généralement à partir de cœfficients stockés dans un bloc mémoire. Ces coefficients doivent donc être redéfinis à chaque fois que la précision de l'opérateur est modifiée. Il faut donc exécuter un travail d'intégration supplémentaire pour insérer ces opérateurs au sein d'une implantation d'algorithme de traitement du signal. Éviter la normalisation permet de rendre l'intégration plus aisée, mais les méthodes classiques d'inversion de la mantisse ne peuvent plus être utilisées.

Dans ce papier, une implantation de l'algorithme de Newton-Raphson pour le calcul d'inverse est présentée. Celle-ci ne nécessite ni normalisation des données ni calcul et stockage de coefficients, ce qui facilite l'intégration dans une architecture matérielle en tant qu'IP et maximise le débit sur FPGA comme nous le verrons par la suite.

La suite de l'article est organisée de la manière suivante. La section 2 présente l'algorithme de Newton-Raphson et ses implantations usuelles. La méthode permettant d'éviter la normalisation est présentée en section 3. Ensuite, l'implantation de l'algorithme de Newton-Raphson est présentée en section 4. La section 5 présente les résultats de cette architecture avant de conclure l'article en section 6.

2 L'algorithme de Newton-Raphson et ses implantations

L'algorithme de Newton-Raphson est une méthode itérative qui permet d'approximer des fonctions linéaires et où une itération double environ la précision de l'approximation. Dans le cas du calcul d'inverse, l'approximation finale x_n de $\frac{1}{a}$ est obtenue après n itérations de l'équation suivante :

$$x_{i+1} = x_i(2 - ax_i) \tag{1}$$

Ces travaux sont financés par le labex CominLabs dans le cadre du projet SABRE.

où x_0 est la première approximation de $\frac{1}{a}$ qui est également une entrée de l'algorithme. Cet algorithme peut être utilisé en représentation flottante ou fixe. La méthode usuelle consiste à considérer que a est dans un intervalle prédéfini comme $[1, 2[$, $[0.5, 1[$ ou $[0, 1[$. Avec cette connaissance sur a, il est possible de déterminer une première approximation ayant une bonne précision [2, 3, 4, 5]. Augmenter la précision initiale permet de diminuer le nombre d'itérations nécessaires mais cette première stratégie demande donc une normalisation des données en entrée et en sortie de l'opérateur. Dans la suite, une méthode alternative pour la représentation en virgule fixe et qui ne demande pas de normalisation est présentée. L'opérateur devient alors directement intégrable dans une architecture sans éléments supplémentaires.

L'architecture présentée dans ce papier utilise de la logique combinatoire pour générer une première approximation de $\frac{1}{a}$. Cette approche est moins précise que les solutions utilisant des cœfficients lus dans un blocs mémoire, et nécessitera éventuellement plus d'itérations. L'avantage de cette solution est la possibilité de montée en fréquence de l'opérateur. En effet, sur les FPGA Xilinx, les cellules DSP ont une fréquence maximale de fonctionnement supérieure aux BRAMs. Dans le cas de la famille Virtex-7, la limite pour les cellules DSP est de 740 MHz alors que celle des BRAMs est de 638.6 MHz [6]. Cette différence de fréquence de fonctionnement est également visible sur les familles de FPGA plus récentes comme le Virtex Ultrascale. En effet, la fréquence proposée par les BRAMs est de 660 MHz, ce qui reste toujours inférieur à celle des cellules DSP (741 MHz) [7]. La solution proposée est donc mieux adaptée pour les applications demandant un fort débit de traitement, en plus d'offrir de la flexibilité et une facilité d'intégration.

3 Éviter la normalisation

3.1 Approximation à un bit de précision

L'idée présentée ici repose sur la condition de convergence de l'algorithme de Newton-Raphson :

$$0 < a \times x_0 < 2. \tag{2}$$

L'objectif ici est de trouver un x_0 adapté à a, alors que la stratégie habituelle consiste à normaliser a pour le ramener dans un intervalle connu et ainsi utiliser une méthode classique d'approximation. Plus le produit $a \times x_0$ est proche de 1 et plus la convergence sera rapide. La valeur à inverser a est au format $uQm.p$, soit une valeur non signée avec m bits pour la partie entière et p bits pour la partie fractionnaire. Ce format est étendu au format $uQn.n$ avec $n = max(m, p)$. L'opérateur présenté ici a donc une précision absolue de 2^{-n} et la sortie x_n sera aussi au format $uQn.n$. En représentation binaire, a s'écrit :

$$a = a_{n-1}a_{n-2} \dots a_0 a_{-1} \dots a_{-n} \tag{3}$$

Soit j l'indice du bit 1 principal, cela implique :

$$2^j \leq a < 2^{j+1}. \tag{4}$$

Une valeur de x_0 qui respecte la condition de convergence peut alors être déduite :

$$x_0 = 2^{-(j+1)}. \tag{5}$$

Les équations (4) et (5) donnent :

$$0.5 \leq a \times x_0 < 1. \tag{6}$$

Cette valeur de x_0 donne donc une approximation de $\frac{1}{a}$ a un seul bit de précision. Ce x_0 peut être facilement obtenu en appliquant un détecteur de bit 1 principal (par la suite appelé LOD pour Leading One Detector) appliqué sur a, et en prenant le symétrique du résultat. Ce LOD peut facilement être réalisé avec une architecture cascadant des portes ET et des portes NON. Si cette solution est très performante pour une implantation ASIC, sur FPGA, il peut être intéressant d'utiliser une autre méthode. Cette alternative comporte trois étapes :

1. Prendre le symétrique de a, noté S(a)
2. Calculer le complément à 2 de S(a)
3. Appliquer un ET bit à bit entre ce complément à 2 et S(a)

Cette méthode est efficace sur FPGA grâce aux blocs de propagation de retenue, utiles pour le complément à 2, ce qui permet d'atteindre une plus grande fréquence de fonctionnement. Le résultat de ces étapes donne exactement $2^{-(j+1)}$, soit la valeur souhaitée pour x_0. En effet, selon (4), a s'écrit :

$$a = 0 \dots 01 a_{j-1} \dots a_{-n}. \tag{7}$$

Ce qui implique la valeur de S(a) :

$$BR(a) = a_{-n} \dots a_{j-1} 10 \dots 0. \tag{8}$$

Et pour celle de son complément à 2 $TC(S(a))$:

$$TC(BR(a)) = \overline{a_{-n}} \dots \overline{a_{j-1}} 10 \dots 0. \tag{9}$$

Finalement, le ET bit à bit entre S(a) et $TC(S(a))$ donne la valeur voulue pour x_0 :

$$a_{-n} \dots a_{j-1} 10 \dots 0 \quad \& \quad \overline{a_{-n}} \dots \overline{a_{j-1}} 10 \dots 0 = 2^{-(j+1)}. \tag{10}$$

Cette méthode ne dépend pas du nombre de bits de l'entrée. Il est donc possible de l'implanter de manière générique dans un langage de description matérielle.

3.2 Approximation à 2 bits de précision

Lors de l'utilisation de l'algorithme de Newton-Raphson, la stratégie habituelle est d'avoir le plus de précision possible pour x_0 [2, 3]. Cela permet de diminuer le nombre d'itérations nécessaires. Il faut donc améliorer l'approximation présentée précédemment. En effet, il est préférable d'utiliser un peu de logique combinatoire plutôt qu'une itération complète pour gagner un bit de précision en plus. Cette amélioration part de la distinction de cas suivante sur a_{j-1}.

Si $a_{j-1} = 1$, alors :

$$1.5 \times 2^j \leq a < 2^{j+1} \quad (11)$$

et donc, si x_0 vaut toujours $2^{-(j+1)}$:

$$0.75 \leq a \times x_0 < 1. \quad (12)$$

Il y a donc déjà deux bits de précision dans ce cas.
A l'inverse, si $a_{j-1} = 0$, alors :

$$2^j \leq a < 1.5 \times 2^j \quad (13)$$

et, pour la même valeur de x_0 :

$$0.5 \leq a \times x_0 < 0.75. \quad (14)$$

La valeur de x_0 peut donc être multipliée par 1.5 pour obtenir les deux bits de précision souhaités, ce qui donne :

$$x_0 = 2^{-(j+1)} + 2^{-(j+2)} \quad (15)$$

et par suite, $0.75 \leq a \times x_0 < 1.125$. La condition sur x_0 à respecter pour avoir la précision souhaitée en est déduite : le bit à l'indice $-(j+2)$ doit être le complément de a_{j-1}. Cette opération demande un cycle supplémentaire et l'ajout d'un bit dans la partie fractionnaire de x_0 pour prévenir les erreurs d'arrondis. Un raisonnement similaire peut être mené pour une précision de 3 bits, avec comme résultat :

$$x_{0_{-(j+1)}} = a_j, x_{0_{-(j+2)}} = \overline{a_{j-1}}, x_{0_{-(j+3)}} = \overline{a_{j-2}}. \quad (16)$$

Ce qui donne : $0.875 \leq a \times x_0 < 1.125$.

Dans la suite, l'approximation à 3 bits de précision est considéré. En effet, le cas d'usage le plus courant dans la bibliographie est l'inverse sur 16 bits. L'approximation à 3 bits de précision permet donc d'assurer cette précision en 3 itérations de l'algorithme. Pour les approximations à 1 ou 2 bits, il faut respectivement 5 et 4 itérations.

4 Architecture globale

La Figure 1 présente l'implantation globale de l'algorithme de Newton-Raphson pour le calcul d'inverse. Le bloc de première approximation (PA) est l'implantation de l'approximation à 3 bits de précision présentée en section 3. Ce bloc transmet la valeur de a et fournit x_0 sur $2n+2$ bits, le tout en 2 cycles d'horloge. L'implantation du bloc LOD est adapté pour toute taille de l'entrée. Les architectures de LOD dans [8, 9] utilisent des blocs élémentaires de LOD sur 2 ou 4 bits. Cela est efficace pour des tailles d'entrées suivant les puissances de 2. La proposition de ce papier concernant ce bloc permet donc d'économiser des ressources pour des tailles d'entrées entre 2 puissances de 2 successives.

Le bloc d'itération de l'algorithme de Newton-Raphson (NR_i) produit la i^{th} évaluation x_i selon l'équation (1). Le bloc calcul d'abord $2 - ax_i$ puis effectue la multiplication du résultat avec x_i. Ces blocs contiennent deux multiplieurs et sont totalement pipelinés. Les bus de données doivent être réduits en cours de process. En effet, les différentes opérations arithmétiques génèrent des bits inutiles pour la précision souhaitées. Elles peuvent donc être ignorées pour éviter de trop contraindre l'architecture. Tous les blocs d'itérations sont identiques sauf le dernier, qui n'a pas à transmettre la valeur de a.

TABLE 1 – Résultats : Inverse 16 bits sur Virtex-7 690T

	Architecture avec PA 1 bit	Architecture avec PA 3 bits
LUTs	159	111
DSP48E1	10	6
Latence (cycles)	31	20
Fréquence max (MHz)	740	740

5 Résultats d'implantation

Cette section est composée de deux parties. Tout d'abord, les résultats d'implantation de l'architecture globale sur le FPGA Virtex-7 690T sont présentés. Ensuite, les performances de cette proposition sont comparées à l'architecture de [4], qui vise également une fréquence de fonctionnement élevée.

5.1 Opérateur de calcul d'inverse

L'opérateur complet est implanté sur cible FPGA. L'implantation est générique, de sorte à ce que le nombre d'itérations et la taille de l'entrée puissent être choisis. L'architecture est totalement pipelinée afin d'atteindre la plus grande fréquence de fonctionnement possible. Le FPGA ciblé est un Virtex-7 690T. Les résultats sont présentés dans la Table 1 et montrent la différence entre une première approximation à 1 bit de précision et une à 3 bits de précision, qui demande un cycle supplémentaire. Comme la précision double environ à chaque itération, la solution avec la PA 1 bit a besoin de 5 itérations, soit 2 de plus que la seconde solution. Cela explique les différences de ressources utilisées. La taille d'entrée considérée ici permet d'utiliser les cellules DSP48E1 qui contiennent des multiplieurs 25x18. Pour des entrées plus grandes, il est possible d'utiliser une combinaison de cellules DSP, ce qui maintiendrait la fréquence maximale de l'architecture. La fréquence ici atteinte est d'ailleurs la plus grandes possible au sein des cellules DSP48E1 totalement pipelinées [10, 6]. Cette optimisation recherchée de la fréquence explique l'ajout d'étages de registres par rapport à l'architecture montrée en Figure 1. Pour les résultats de la Table 1, chaque bloc d'itération possède 6 niveaux de pipeline. L'implantation est générique et facilement portable à toutes cibles FPGA et à toutes tailles d'entrée.

5.2 Comparaison à l'état de l'art

L'architecture sans normalisation est comparée avec la solution de [4], qui vise aussi une haute fréquence de fonctionnement sur un FPGA Virtex-4 SX35-12 en utilisant une première

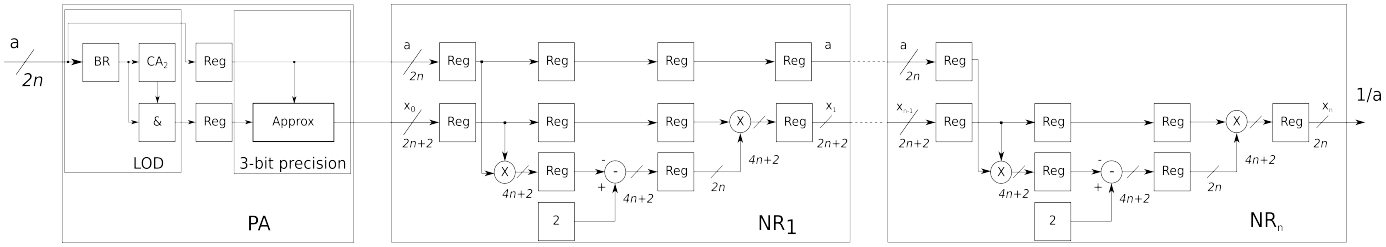

FIGURE 1 – Architecture globale de l'opérateur de calcul d'inverse

TABLE 2 – Résultats : Division 16 bits sur Virtex-4 SX35

	Architecture avec PA 3 bits	[4]
Slices	347	1478
LUTs	372	2091
Flip-Flop	568	1820
DSP48E1	7	7
Latence (cycles)	25	112
Normalisation	Non	Oui
Stockage de cœfficients	Non	Oui (34Kb)
Fréquence max (MHz)	294.1	294.1

approximation polynomiale. Cette approximation est assez précise, et seulement deux itérations sont nécessaires par la suite. Pour la comparaison, la solution proposée dans ce papier a été portée sur la même cible. Un multiplieur et un registre à décalage ont été ajoutés afin d'en faire un diviseur. Les résultats sont présentés dans la Table 2. La fréquence atteinte dans les deux cas, 294.1 MHz sert de point de comparaison. La proposition sans normalisation utilise donc 82% de LUTs et 69% de Flip-Flop en moins. La diminution de cycles de latence est également significative. Comme il n'y a pas de valeurs stockées dans l'architecture, elle est plus flexible et plus facilement intégrable en tant qu'IP.

6 Conclusion

Ce papier présente une implantation en virgule fixe de l'algorithme de Newton-Raphson. Il a été prouvé qu'il est possible d'aboutir à une architecture qui ne demande pas de normalisation des données. Cela permet d'avoir une IP prête à être utilisée. Cette adaptation est permise par l'utilisation d'une technique à base de LOD pour la première approximation. Cela permet d'être générique par rapport à la taille des entrées. Ce LOD peut être réalisé à l'aide d'un complément à 2 pour bénéficier des blocs de propagation de retenue et ainsi assurer la montée en fréquence sur FPGA. L'architecture finale nécessite peu de ressources, est totalement pipelinée et n'a pas besoin de bloc mémoire pour stocker des cœfficients. Il s'agit donc d'un opérateur optimisé pour une cible FPGA qui peut directement être intégré dans n'importe quelle architecture de traitement du signal, peu importe le format en virgule fixe utilisé.

Références

[1] S. F. Obermann and M. J. Flynn. Division algorithms and implementations. *IEEE Transactions on Computers*, 46(8) :833–854, August 1997.

[2] A. Rodriguez-Garcia, L. Pizano-Escalante, R. Parra-Michel, O. Longoria-Gandara, and J. Cortez. Fast fixed-point divider based on Newton-Raphson method and piecewise polynomial approximation. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–6, December 2013.

[3] H. C. Neto and M. P. Vestias. Very low resource table-based FPGA evaluation of elementary functions. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–6, December 2013.

[4] M. P. Vestias and H. C. Neto. Revisiting the Newton-Raphson Iterative Method for Decimal Division. In *2011 International Conference on Field Programmable Logic and Applications (FPL)*, pages 138–143, September 2011.

[5] M. Ito, N. Takagi, and S. Yajima. Efficient initial approximation for multiplicative division and square root by a multiplication with operand modification. *IEEE Transactions on Computers*, 46(4) :495–498, April 1997.

[6] Xilinx. Virtex-7 t and xt fpgas data sheet : Dc and ac switching characteristics, 2016 (Accessed : 2017-02-10).

[7] Xilinx. Virtex ultrascale fpgas data sheet : Dc and ac switching characteristics, 2016 (Accessed : 2017-02-10).

[8] K. Kunaraj and R. Seshasayanan. Leading one detectors and leading one position detectors - An evolutionary design methodology. *Canadian Journal of Electrical and Computer Engineering*, 36(3) :103–110, 2013.

[9] K. H. Abed and R. E. Siferd. VLSI Implementations of Low-Power Leading-One Detector Circuits. In *Proceedings of the IEEE SoutheastCon, 2006*, pages 279–284, March 2006.

[10] Xilinx. 7 series dsp48e1 slice-user guide, 2016 (Accessed : 2017-02-10).