Évaluation polynomiale en-ligne de fonctions élémentaires sur FPGA

Jean-Luc Beuchat, Arnaud Tisserand

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Évaluation polynomiale en-ligne de fonctions élémentaires sur FPGA

Jean-Luc Beuchat et Arnaud Tisserand

N° 4557

Septembre 2002

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4557--FR

Évaluation polynomiale en-ligne de fonctions élémentaires sur FPGA

Jean-Luc Beuchat et Arnaud Tisserand

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arénaire

Rapport de recherche n° 4557 — Septembre 2002 — 16 pages

Résumé : Cet article présente une architecture matérielle pour l'évaluation de fonctions élémentaires en arithmétique en-ligne sur circuits FPGA. Les points particuliers traités sont la détermination automatique de « bons » polynômes d'approximation, et la génération automatique de la description VHDL synthétisables des opérateurs correspondants. Cette approche a été implantée et validée sur des circuits FPGA Xilinx.

Mots-clés : arithmétique des ordinateurs, évaluation des fonctions élémentaires, approximation polynomiale, arithmétique en-ligne, circuits FPGA.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

On-Line Polynomial Evaluation of Elementary Functions on FPGA

Abstract: This article presents a hardware architecture for the evaluation of some elementary functions using on-line arithmetic on FPGA circuits. The main contributions are the automatic determination of "accurate" polynomial approximations and the automatic generation of the corresponding synthesizable VHDL operators. This approach has been implemented and validated on Xilinx FPGAs.

Key-words: computer arithmetic, elementary function evaluation, polynomial approximation, on-line arithmetic, FPGA circuits.

1 Introduction

Lors de la conception d'opérateurs arithmétiques, deux modes de transmission des nombres s'affrontent : le mode parallèle et le mode sériel. La majeure partie des circuits de calcul utilise le mode parallèle : processeurs généraux, DSP, cartes graphiques... Toutefois, dans certaines applications où la surface de silicium est un paramètre critique, une arithmétique sérielle peut être utilisée avantageusement. En effet, les opérateurs sériels sont souvent plus petits que leurs équivalents parallèles. La faible vitesse de calcul due au caractère séquentiel de cette arithmétique peut être contre-balancée par le parallélisme introduit par le pipeline au niveau du chiffre lors d'opérations successives. On a alors une décomposition des opérations en tout petits blocs comme dans le pipeline des processeurs récents pour en augmenter le débit.

En arithmétique sérielle, les nombres peuvent être transmis soit avec les poids forts en tête, soit avec les poids faibles en tête. L'arithmétique sérielle classique, ou « *Least Significant Digit First* », abrégée LSDF, parfois appelée arithmétique bit-série, permet de réaliser facilement l'addition et la multiplication. Par contre, les opérations de comparaison et division ne sont pas possibles directement, c'est-à-dire avant d'avoir reçu la totalité des chiffres des opérandes. L'arithmétique sérielle classique a été utilisée principalement pour l'implantation d'algorithmes simples de traitement du signal, comme des filtres numériques [10]. Il existe des extensions à des bases supérieures à 2, ce qui donne l'arithmétique chiffre-série [15, 1]. Cette augmentation de la base permet de diminuer le nombre de cycles pour effectuer les calculs mais conduit à des étages de calcul plus complexes, ce qui peut globalement ralentir le circuit.

L'arithmétique en-ligne ou « *Most Significant Digit First* », abégrée MSDF, est une arithmétique sérielle où les données circulent les poids forts en tête dans les opérateurs. Cette arithmétique, basée sur un système redondant d'écriture des nombres, permet d'effectuer toutes les opérations habituelles avec les poids forts en tête. L'arithmétique en-ligne a été utilisée avec succès pour des circuits de traitement du signal [16], les réseaux de neurones artificiels [9], le contrôle numérique de micro-systèmes [6] ou encore la recherche de séquences dans des chaînes en bio-informatique [12]. Toutefois, les opérations utilisées dans ces applications sont assez simples comme des additions/soustractions et des multiplications et, plus rarement, des fonctions algébriques comme la division ou la racine carrée. L'évaluation des fonctions élémentaires ($\sin$, $\cos$, $\exp$, $\log \ldots$) en arithmétique en-ligne a été rarement étudiée jusqu'au niveau de l'implantation réelle en matériel.

Nous étudions dans cet article l'implantation d'approximations polynomiales pour évaluer des fonctions élémentaires en arithmétique en-ligne. La précision ciblée est celle requise par les applications classiques de traitement du signal et de contrôle numérique, soit d'une dizaine à une trentaine de bits. Dans certaines applications, il peut être nécessaire d'évaluer non pas une mais plusieurs fonctions élémentaires. Outre l'argument x, la fonction f doit aussi être spécifiée en entrée de l'opérateur. L'intérêt est de pouvoir évaluer plusieurs fonctions élémentaires successivement, typiquement de 2 à 6 fonctions différentes, sans devoir modifier le circuit. Nous implantons les architectures correspondantes sur des circuits FPGA de la famille Virtex de Xilinx. Les « *Field Progammable Gate Arrays* », ou FPGA en abrégé, sont des circuits programmables dont les dernières générations permettent d'envisager la conception de grands systèmes utilisant des blocs de calculs complexes comme des fonctions élémentaires. Les résultats obtenus sont transposables aux circuits intégrés numériques ou à d'autres familles de FPGA. Ce travail est constitué de deux parties principales. La première est l'automatisation de la recherche d'une « bonne » approximation polynomiale une fois les spécifications des entrées et du résultat données. La seconde partie est la génération automatique du code VHDL synthétisable pour implanter l'architecture de calcul.

Cet article débute en section 2 par une courte présentation de l'arithmétique en-ligne et des représentations des nombres utilisées. La détermination automatique de polynômes d'approximation est abordée à la section 3. La section 4 présente l'architecture de calcul et sa génération automatique en VHDL. Les résultats obtenus sont présentés en section 5 et comparés avec quelques autres techniques d'évaluation de fonctions élémentaires. Enfin, nous donnons quelques conclusions sur ce travail et des pistes de recherche pour de futurs travaux en section 6.

2 Arithmétique en-ligne et évaluation des fonctions élémentaires

Dans cette section, nous rappelons quelques notions d'arithmétique des ordinateurs. Dans un premier temps, nous donnons une brève introduction à l'arithmétique en-ligne et aux opérations de base utilisées dans

la suite, à savoir l'addition, la multiplication et des opérations dérivées. Enfin, nous présentons et justifions la méthode d'évaluation des fonctions élémentaires proposée.

2.1 Arithmétique en-ligne

L'arithmétique en-ligne, proposée par Trivedi et Ercegovac en 1977 [19], est une arithmétique sérielle où les chiffres arrivent séquentiellement avec les poids forts en tête. Cette méthode de calcul n'est possible que par l'utilisation d'un système redondant d'écriture des nombres qui limite les propagations de retenues. En effet, dans un tel système, toutes les opérations usuelles peuvent être effectuées avec les poids forts en tête. Sans cette notation redondante, l'addition provoque des progapations de retenues désastreuses en mode MSDF. Ceci est le principal avantage par rapport à l'arithmétique sérielle classique, en mode LSDF, qui ne permet pas de débuter le calcul d'opérations comme la division sans avoir reçu tous les chiffres des opérandes.

Les systèmes redondants d'écriture des nombres autorisent plusieurs écritures distinctes de certains nombres. Ceci permet à l'aide d'astucieux codages de limiter les propagations de retenues lors de l'addition. Le temps nécessaire pour effectuer l'addition devient alors indépendant de la longueur des opérandes. En pratique, on utilise un ensemble de chiffres plus grand que celui qui serait nécessaire pour un système d'écriture non redondant. Par exemple, en base r alors que l'ensemble de chiffres pour le système non redondant est $\mathcal{D} = \{0,1,\ldots,r-1\}$, celui de l'un des systèmes redondants possibles est $\mathcal{D}_\rho = \{-\rho,\ldots,\rho\}$ où $2\rho+1 \geq r$ et r une puissance de deux. Cette représentation avec un ensemble de chiffres signés a été proposée par Avizienis en 1961 dans [2]. Il existe dans la littérature différents travaux sur l'utilisation de bases et d'ensembles de chiffres plus complexes, mais ils sont inutilisables en pratique dans les circuits intégrés.

Le choix de la base est une question délicate *a priori*. Tout est affaire de compromis entre différents paramètres comme la base, le codage des chiffres et les algorithmes utilisés. En pratique, les petites bases comme 2 et 4 sont souvent bien « meilleures » que des bases plus grandes. En effet, augmenter la base permet de réduire le nombre de cycles d'horloge, mais cela se paye au niveau des étages de calcul qui sont alors plus complexes.

Dans la suite, nous utiliserons la base 2. Il faut encore choisir un codage des chiffres adapté aux circuits FPGA. Nous adoptons la représentation *borrow-save*, clairement illustrée dans [3], où un chiffre x_i appartenant à $\mathcal{D}_\rho = \{-1,0,1\}$ est codé à l'aide d'un bit positif x_i^+ et d'un bit négatif x_i^-, tels que $x_i = x_i^+ - x_i^-$. D'autres codages sont possibles, mais celui-ci permet un traitement simple et naturel des nombres signés sans devoir passer par un artifice comme le complément à 2.

En général, chaque opérateur réalise une seule opération de base. Ensuite, la composition de calculs plus complexes se fait simplement en chaînant différents opérateurs successivement (figure 1a). Outre l'opération effectuée, un opérateur en-ligne est caractérisé par les principaux paramètres suivants :

- Le délai en-ligne ou latence, noté δ dans la suite, est défini comme la différence des rangs des chiffres entrant et sortant (figure 1b). C'est-à-dire qu'il faut entrer $i+\delta$ chiffres des opérandes pour produire le $i$ème chiffre du résultat. Le délai en-ligne dépend de l'algorithme utilisé et du système de numération choisi. En pratique, cette latence est toute petite (de 1 à 4 cycles pour les opérations de base).
- Le temps de cycle ou période, noté τ dans la suite, correspond au plus grand temps de propagation du signal dans l'opérateur (figure 1b). C'est cette valeur qui va borner la fréquence maximale d'utilisation du circuit.
- Enfin la surface de circuit utilisée dépend de la complexité de l'algorithme mis en œuvre et des tailles des opérandes. Dans le cas des FPGA, nous mesurons la surface par le nombre de blocs logiques ou CLB (pour *Configurable Logic Block*).

Les paramètres δ et τ sont étroitement liés et une augmentation de l'un entraîne généralement une diminution de l'autre. Lors de la conception, une des difficultés consiste à trouver le bon compromis entre le délai en-ligne et le temps de cycle.

Dans la suite, les principales opérations en-ligne utilisées seront l'addition et la multiplication. On trouve dans [4], par exemple, toutes les informations sur la conception des opérateurs correspondants. Pour information, nous présentons dans la table 1 la taille et la vitesse de quelques opérateurs pour différentes tailles des opérandes. Ces données sont issues de résultats de synthèse d'une bibliothèque interne d'opérateurs en-ligne sur un FPGA Virtex de type XCV50E-6 avec Synplify. Les additionneurs en-ligne ont une taille indépendante de la taille des opérandes. Les opérateurs comme le multiplieur-additionneur $XY+Z$ ou le binômieur $aX+Y$ (où a est une constante) sont dérivés facilement de la multiplication et sont très utiles pour l'évaluation de

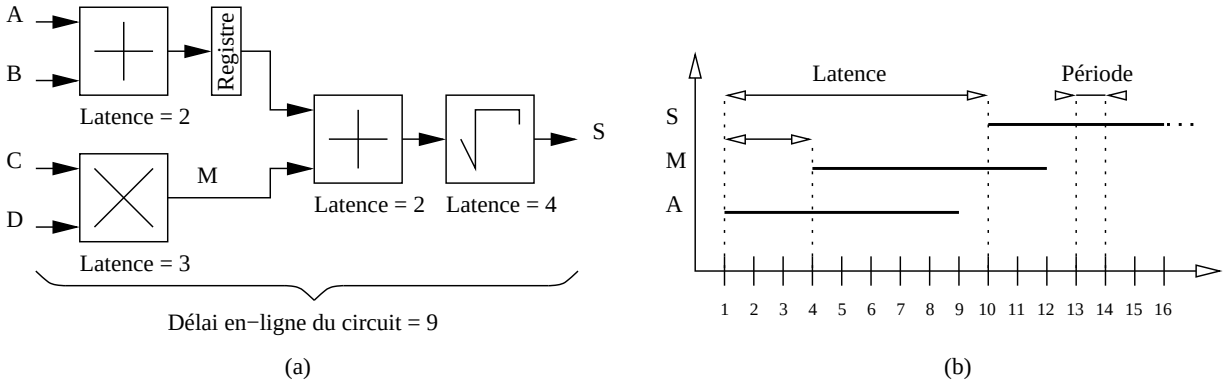

FIG. 1 – *Délai en-ligne et temps de cycle des opérateurs en-ligne.*

opération → ↓ taille	$X+Y$ $\delta=2$	$X+Y+Z$ $\delta=3$	XY $\delta=3$	$aX+Y$ $\delta=2$	$XY+Z$ $\delta=3$
8	2 CLB 5,2 ns	3 CLB 7,4 ns	24 CLB 12,4 ns	17 CLB 9,3 ns	25 CLB 12,2 ns
16	2 CLB 5,2 ns	3 CLB 7,4 ns	52 CLB 12,4 ns	23 CLB 9,4 ns	65 CLB 12,7 ns
24	2 CLB 5,2 ns	3 CLB 7,4 ns	77 CLB 12,1 ns	29 CLB 9,6 ns	102 CLB 13,1 ns
32	2 CLB 5,2 ns	3 CLB 7,4 ns	121 CLB 12,8 ns	36 CLB 9 ns	132 CLB 13,2 ns

TAB. 1 – *Taille et vitesse des principaux opérateurs en-lignes.*

polynômes. En arithmétique en-ligne, il est très facile de modifier un opérateur effectuant le produit XY en un opérateur effectuant le calcul $XY+Z$. En effet, on trouve un étage d'addition à la fin du multiplieur standard. On peut alors facilement lui fusionner un autre petit additionneur pour prendre un opérande supplémentaire. L'intérêt est limiter le délai en-ligne de l'opérateur dérivé. Un opérateur calculant $XY+Z$ a une latence de 3 alors qu'une composition multiplieur puis additionneur donne une latence de 5 cycles dont 3 pour le multiplieur et 2 pour l'additionneur. Ceci se fait généralement sans augmenter le temps de cycle de l'opérateur résultant.

2.2 Évaluation des fonctions élémentaires en-ligne

S'il existe des méthodes de conception d'opérateurs en-ligne destinés aux fonctions arithmétiques et algébriques [7, 4], peu de chercheurs proposent des architectures dédiées à l'évaluation des fonctions élémentaires ($\sin$, $\cos$, $\exp$, $\log\ldots$). Ces dernières jouent cependant un rôle important dans le calcul scientifique et la réalisation d'opérateurs les implantant efficacement nous semble indispensable à l'essor de l'arithmétique en-ligne.

L'évaluation des fonctions élémentaires peut se faire à l'aide de différents types d'algorithmes. On compte essentiellement trois grands classes de méthodes : les approximations polynomiales ou rationnelles, les algorithmes à base d'additions et de décalages et enfin les méthodes à base de tables. Il existe aussi des mélanges de ces trois principaux types de méthodes. Toutes ces méthodes sont détaillées dans [13].

Les méthodes à base de tables reposent sur l'utilisation de petites tables et d'un petit nombre d'opérations très simples comme des additions. Ces méthodes sont limitées aux petites précisions, jusqu'à une vingtaine de bits, même si certaines améliorations récentes semblent prometteuses [5]. Toutefois, les méthodes à base de tables ne sont pas intéressantes avec une arithmétique sérielle car on n'utilise pas facilement le fait que les données arrivent chiffre par chiffre. On pourrait utiliser le caractère sériel des entrées pour réaliser le décodage des adresses avec un temps de cycle plus petit, le temps de passage dans un seul multiplexeur plutôt que celui

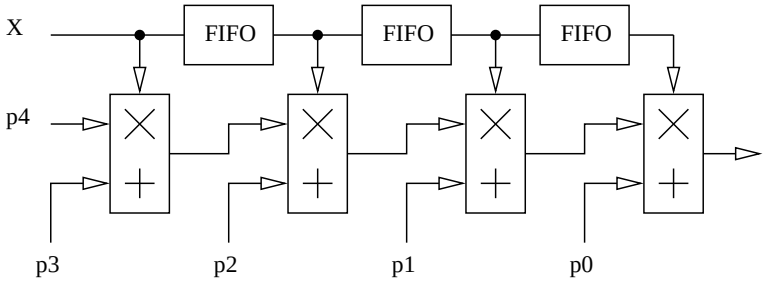

FIG. 2 – *Architecture d'un polynomieur en-ligne.*

de plusieurs en arbre. Mais la taille du mot d'adresse est suffisamment faible en pratique pour qu'il ne soit pas utile de pipeliner son décodage. De plus, les données contenues dans la table sont facilement compressibles par les optimiseurs logiques des outils de synthèse. Par exemple, dans [5], on trouve des facteurs de compression de 19 entre le nombre total de bits de la table et le nombre de bits utilisés réellement dans le FPGA. Pour des raisons de taille des tables générées, il n'est pas possible de se servir du caractère sériel des entrées.

Les algorithmes à base d'additions et de décalages, comme le célèbre algorithme CORDIC, fournissent un chiffre du résultat à chaque cycle mais nécessitent de connaître quelques chiffres des opérandes dès le début du calcul. Il est possible de modifier ces algorithmes pour obtenir un comportement totalement sériel mais le surcoût matériel est important ce qui fait perdre l'avantage principal de ces méthodes qui est la petite taille des opérateurs. On trouve plusieurs essais de mise en œuvre d'algorithmes à base d'additions et de décalages pour l'évaluation de certaines fonctions élémentaires en-ligne [18, 14]. Les résultats obtenus montrent que les algorithmes deviennent assez complexes. De plus, les algorithmes proposés ne fonctionnent que pour certaines fonctions bien particulières.

Il reste les méthodes à base d'approximations polynomiales ou rationnelles pour évaluer les fonctions élémentaires en arithmétique en-ligne, ou même en arithmétique bit-série traditionnelle. Les approximations basées sur l'évaluation d'une fraction rationnelle permettent d'obtenir une grande précision mais nécessitent d'effectuer une division très coûteuse en matériel. Étant donnée la précision ciblée pour nos applications, nous ne nous intéresserons qu'au cas des approximations polynomiales. En écrivant le polynôme d'approximation sous forme de Horner, on peut obtenir un opérateur sériel particulièrement régulier où le même élément de calcul $XY + Z$ se répète sur les différents étages. On a par exemple :

$$p(x) = p_0 + p_1 x + p_2 x^2 + p_3 x^3 + p_4 x^4 = p_0 + \Big(p_1 + \big(p_2 + (p_3 + p_4 x)x\big)x \Big) x.$$

La figure 2 présente l'architecture d'un polynomieur, opérateur qui évalue ici un polynôme de degré 4, sous forme de Horner. Les FIFO permettent de retarder l'arrivée du premier chiffre pour les étages suivants pour compenser le délai en-ligne des opérateurs. La synchronisation des différents étages est, dans ce cas, triviale. Le délai en-ligne de cette architecture est le produit du degré du polynôme par le délai en-ligne d'un étage qui effectue une opération du type $XY + Z$. Les étages de calcul ont une latence de 3 dans notre cas, on a donc un délai en-ligne global de $3d$. Pour diminuer ce délai, on peut penser à utiliser une méthode de type *divide and conquer*, mais la surface du circuit doit être plus importante pour pouvoir générer les valeurs x^2, x^4…

L'utilisation d'un seul polynôme d'approximation pour évaluer une fonction élémentaire n'est pas nécessairement efficace. En effet, pour obtenir des approximations précises dans un vaste domaine, il faut utiliser un polynôme de degré élevé. En pratique, on préfère découper le domaine de départ en plusieurs intervalles et utiliser un polynôme de plus petit degré sur chaque intervalle. L'idée est donc de lire dans de petites tables les coefficients des polynômes à utiliser en fonction des premiers chiffres de l'argument. Ceci est très efficace en arithmétique en-ligne car les poids forts permettent de découper trivialement le domaine de départ en petits intervalles. C'est cette solution nous allons détailler dans la suite. Dans sa thèse [11], Kla propose une méthode de calcul similaire par lecture de table suivie d'une évaluation de polynôme, mais il ne donne aucun détail sur

sa mise en œuvre pratique. Il définit tout d'abord l'opérande x par

$$x = \underbrace{\sum_{i=1}^{n} x_i 2^{-i}}_{x[n]} + 2^{-n} \underbrace{\sum_{i=1}^{\infty} x_{n+i} 2^{-i}}_{\tilde{x}},$$

puis établit le développement de Taylor de la fonction à évaluer $f(x)$ autour de $x[n]$:

$$\begin{aligned} f(x) &= f\left(x[n] + 2^{-n}\tilde{x}\right) \\ &= f(x[n]) + 2^{-n} \underbrace{\sum_{i=1}^{m} f^{(i)}(x[n]) \frac{2^{-(i-1)n}\tilde{x}^i}{i!}}_{q(\tilde{x})} + \epsilon. \end{aligned}$$

Le principe consiste à lire les premiers chiffres de $f(x[n])$ dans une table, puis à calculer $q(\tilde{x})$ à l'aide d'un polynômieur. La valeur de $f(x[n])$ est néanmoins entachée d'une erreur dont il faut tenir compte lors de l'évaluation de $q(\tilde{x})$. Kla n'a toutefois jamais implanté son algorithme et ne fournit aucune estimation de la taille du circuit et de la précision obtenue. De plus, nous améliorons sa solution en utilisant de biens meilleures approximations.

3 Détermination des approximations polynômiales

En règle générale, l'évaluation d'une fonction élémentaire se fait en deux étapes : la réduction d'argument puis l'évaluation proprement dite [13]. La phase de réduction d'argument permet de se ramener à un petit domaine dans lequel l'approximation utilisée est suffisamment précise. Dans le cas de certaines fonctions comme sinus ou cosinus, on essaye souvent de se ramener dans l'intervalle $[-\pi/4,\pi/4]$ avec des additions/soustractions et des comparaisons. Puis une fois dans cet intervalle, on évalue la bonne fonction grâce aux nombreuses formules de trigonométrie. La phase de réduction d'argument est importante car il n'existe pas de méthode d'évaluation qui soit à la fois très précise et valide dans un grand domaine.

3.1 Réduction d'argument par découpage

Les méthodes classiques de réduction d'argument sont assez complexes en terme de contrôle, comparaisons multiples et traitement de cas particuliers. En arithmétique en-ligne, la comparaison a une latence variable. Nous souhaitons conserver un pipeline régulier au niveau du chiffre. Les chiffres de poids forts arrivant en premier en arithmétique en-ligne, nous pouvons trivialement découper le domaine en intervalles indexés par ces premiers chiffres. En effet, à chaque cycle d'horloge, le nouveau chiffre qui arrive nous renseigne un peu plus sur la position de la valeur finale comme illustré par la figure 3 pour des nombres fractionnaires. Sur cette figure les zones grisées indiquent pour chaque chiffre possible de $\{-1,0,1\}$, en base 2, les intervalles possibles pour la valeur finale. L'arithmétique en-ligne présente la propriété que les valeurs se précisent au fur et à mesure des chiffres qui arrivent. La réduction d'argument se fait en attendant les m premiers chiffres de l'entrée x. Ces premiers chiffres sélectionnent un intervalle possible pour la valeur de x, et il faut alors être capable de faire l'approximation polynomiale dans chaque intervalle. Pour cela on stocke dans une petite table les coefficients du polynôme d'approximation pour chaque intervalle. Cette table est adressée par une écriture non redondante des m premiers chiffres de x. On compte sur l'efficacité des optimiseurs logiques, qui se vérifie en pratique, pour n'implanter que de petites tables en comprimant fortement leur contenu.

Nous présentons dans la table 2, comme exemple pour la fonction exponentielle, la précision de l'approximation théorique obtenue pour différents compromis entre le degré d du meilleur polynôme minimax (présenté ci-dessous) pour l'intervalle d'évaluation I. La précision donnée est le nombre de bits corrects de l'évaluation pour chaque couple (I,d) en ne tenant compte que de l'erreur d'approximation. C'est-à-dire en considérant que l'évaluation s'effectue en précision arbitraire. Il n'y a donc que l'erreur de méthode et pas d'erreur de calcul, dont il faut tenir compte en pratique. Nous reviendrons sur ce point en 5.4. Ce type de réduction d'argument pour l'exponentielle est utilisé, avec d'autres valeurs de découpage, du fait de l'indentité $e^{c+x} = e^c e^x$.

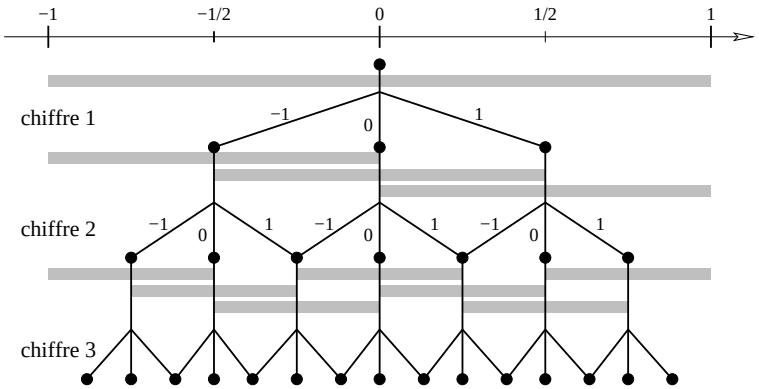

FIG. 3 – *Découpage du domaine avec les premiers chiffres de poids forts.*

$\downarrow d$ $I \rightarrow$	[0,1]	[0,1/2]	[0,1/4]	[0,1/8]	[0,1/16]
1	3,2	5,6	7,8	9,9	11,9
2	6,8	10,2	13,4	16,4	19,5
3	10,8	15,2	19,4	23,4	27,5
4	15,1	20,5	25,7	30,8	35,8
5	19,7	26,1	32,3	38,4	44,4

TAB. 2 – *Compromis découpages/précision obtenue pour e^x (nb. bits justes).*

3.2 Polynômes minimax

Nous devons maintenant trouver un polynôme p qui approche la fonction f sur l'intervalle $I = [a,b]$ déterminé lors de la précédente phase de réduction d'argument. Il existe différentes solutions, comme faire un développement de Taylor autour du centre de l'intervalle, mais ce n'est une bonne approximation qu'autour de ce point et pas globalement sur l'intervalle complet [13].

Une bien meilleure solution consiste à utiliser une approximation au sens de la norme $||f(x) - p(x)||_\infty = \sup_I |w(x)(f(x) - p(x))|$. Cette approximation, appelée *minimax*, se calcule numériquement avec un algorithme dû à Remez [17]. La fonction $w(x)$ est une fonction de poids non nulle, on prend par exemple $w(x) = 1/f(x)$ pour minimiser l'erreur relative.

Un logiciel comme Maple est capable de calculer des approximations minimax. La bibliothèque `numaprox` de Maple fournit la fonction `minimax(f,I,[d,e],w,err)` où d et e sont respectivement le degré du numérateur et celui du dénominateur de la fraction rationnelle recherchée (dans le cas d'un polynôme on fixe $e = 0$). Le calcul qui se fait numériquement est assez sensible à la précision des calculs intermédiaires et peut être instable pour de petites précisions. Il peut donc être nécessaire de calculer avec une précision interne plus grande que celle par défaut. La variable `Digits` de Maple spécifie le nombre de chiffres de base 10 des calculs internes.

Par exemple, pour obtenir le polynôme minimax de degré 3 pour la fonction e^x avec x dans $I = [1/2,1]$, on utilise la commande :

```
minimax(exp(x),x=1/2..1,[3,0],1,'err');
```

ce qui donne le résultat :

$$0{,}9781955669 + \big(1{,}119131954 + (0{,}2664209097 + 0{,}3544901686x)x\big)x.$$

L'erreur d'approximation théorique, contenue dans la variable `err` passée dans le dernier paramètre de la commande, est alors de $0{,}43.10^{-4}$ soit 14,4 bits de précision environ. L'avant dernier paramètre fixé à 1 est la fonction de poids $w(x)$ introduite précédemment: ici on minimise l'erreur absolue.

La précision retournée par Maple à travers la variable `err` est la valeur de $||f(x) - p(x)||_\infty$ sur I où p est le polynôme résultat. Cette erreur est donc seulement l'erreur de l'approximation de f par p indépendamment

de toute évaluation pratique du polynôme. Comme l'évaluation pratique du polynôme p se fait en précision bornée, il faut aussi tenir compte de l'erreur de calcul. Il faut donc choisir un polynôme dont l'erreur d'approximation soit bien plus petite que la précision cible.

De plus, le résultat donné par la procédure `minimax` donne des polynômes avec des coefficients qui sont bien plus précis que ce qui est pratiquement utilisable dans nos opérateurs. En fait, Maple retourne des coefficients au format des calculs intermédiaires. Il faut donc arrondir les coefficients sur le format cible. Nous avons réalisé une procédure Maple qui détermine le meilleur polynôme d'approximation avec des coefficients rationnels au format final. Cette procédure part du polynôme avec des coefficients réels et détermine pour chaque combinaison possible des coefficients et des modes d'arrondi l'erreur d'approximation et retourne celui qui conduit à la plus petite erreur d'approximation.

Par exemple, considérons la fonction exponentielle sur l'intervalle $[1/2,1]$ avec un polynôme de degré 2, une fonction de poids unitaire et des nombres de 10 bits dont 1 en partie entière. Le polynôme minimax théorique trouvé est :

$$1{,}116018832 + 0{,}535470996x + 1{,}065407154x^2$$

pour une erreur d'approximation conduisant à une précision de 9,5 bits environ.

En arrondissant au plus près les 3 coefficients de ce polynôme on a alors:

$$\frac{571}{512} + \frac{137}{256}x + \frac{545}{512}x^2.$$

et une précision de l'approximation de 8,1 bits.

Après avoir testé tous les modes d'arrondi possibles pour chacun des coefficients, on trouve que le meilleur polynôme est :

$$\frac{571}{512} + \frac{275}{512}x + \frac{545}{512}x^2$$

pour une précision de l'approximation de 9,3 bits. Les coefficients ont été arrondi vers le bas pour le premier et le dernier et vers le haut pour le second dans ce cas particulier.

Les valeurs binaires finales à stocker dans la table pour notre exemple sont donc :

$$\begin{aligned} p_0 &= 1{,}000111011; \\ p_1 &= 0{,}100010011; \\ p_2 &= 1{,}000100001. \end{aligned}$$

Cette différence de précision entre les diverses solutions pour arrondir les coefficients oblige à une recherche exhaustive. Le degré des polynômes utilisés en pratique est assez faible, pas plus de 5 pour nos applications. La détermination du meilleur polynôme d'approximation avec les coefficients arrondis ne dure alors que quelques minutes au total.

4 Architecture du système

Notre méthode d'évaluation de fonctions élémentaires nécessite le calcul en-ligne d'un polynôme $p(x)$. Étudions plus en détail la réalisation d'un polynômieur exploitant le schéma de Horner (figure 4). Le premier étage de l'opérateur calculant $p_d x + p_{d-1}$, nous utilisons un binômieur traitant p_d comme une constante en complément à deux [4]. Tous les autres coefficients sont représentés en signe et valeur absolue afin de limiter la surface nécessaire à leur mémorisation. Un module effectue la conversion en borrow-save avant de les transmettre aux binômieurs. Considérons un nombre purement fractionnaire de $n+1$ bits codé en signe et valeur absolue (x_0 dénote le bit de signe). Comme

$$x = (1-2x_0)\cdot\sum_{i=1}^{n} x_i 2^{-i} = (\bar{x}_0 - x_0)\cdot\sum_{i=1}^{n} x_i 2^{-i} = \sum_{i=1}^{n}(\bar{x}_0 x_i - x_0 x_i)2^{-i},$$

il suffit de définir $y_i^+ = \bar{x}_0 x_i$ et $y_i^- = x_0 x_i$ afin d'effectuer la conversion en borrow-save. Cette opération nécessite simplement une bascule mémorisant le bit de signe, ainsi que deux LUT (pour *Look Up Table*) responsables du calcul de y_i^+ et y_i^-.

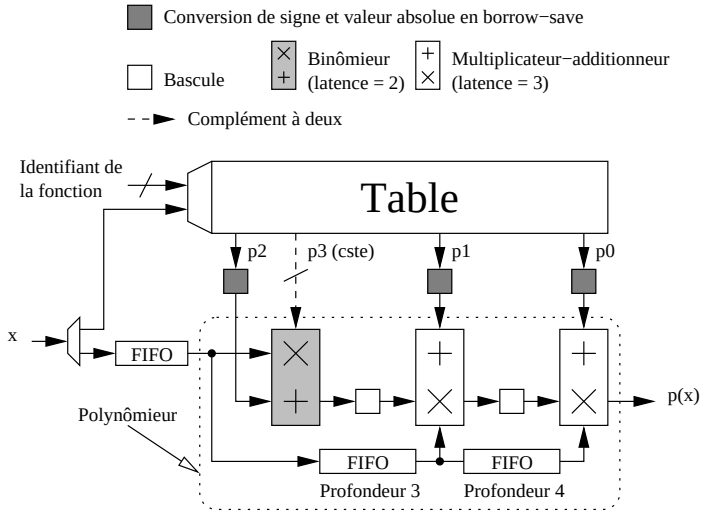

FIG. 4 – *Architecture d'un polynômieur en-ligne de degré trois.*

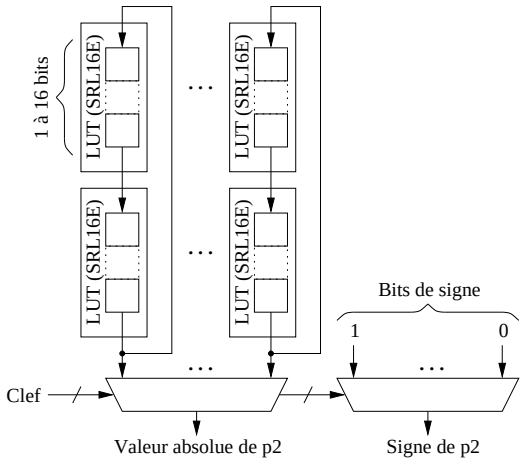

FIG. 5 – *Utilisation des LUT d'un circuit Virtex ou Spartan-II comme des registres à décalage afin de mémoriser les coefficients.*

Les coefficients sont déterminés par une clef constituée des m chiffres de poids forts de x ainsi que de quelques bits spécifiant la fonction désirée. Il est par conséquent nécessaire de placer un FIFO de profondeur m à l'entrée du polynômieur afin de synchroniser l'opérande x et les termes p_i. Différentes alternatives s'offrent à nous pour la réalisation matérielle des tables. La première consiste à mémoriser tous les coefficients dans un unique tableau. Une fois l'intervalle dans lequel se trouve x déterminé, les termes p_i requis sont transférés dans des registres parallèle-sériel, puis transmis chiffre de poids fort en tête au polynômieur. Une seconde approche possible est l'utilisation des LUT des circuits Virtex-E ou Spartan-II en registres à décalage (figure 5).

L'utilisation du système s'avère très simple. À chaque ligne de données du polynô-mieur est associé un signal indiquant la validité des informations transmises. Afin de permettre l'initialisation des opérateurs, ce dernier possède un cycle d'horloge d'avance sur les opérandes [4].

n	d	τ [ns]	nb CLB	Temps [sec]
10	3	14	140 (6%)	171
	4	13	222 (9%)	227
	5	14	290 (12%)	318
12	3	14	162 (7%)	199
	4	14	249 (11%)	249
	5	14	330 (14%)	332
14	3	13	182 (8%)	218
	4	14	262 (11%)	281
	5	15	366 (16%)	351
16	3	14	196 (8%)	222
	4	15	301 (13%)	304
	5	15	396 (17%)	387
18	3	14	221 (9%)	256
	4	14	331 (14%)	305
	5	15	430 (18%)	551
20	3	13	237 (10%)	250
	4	13	363 (15%)	326
	5	14	466 (20%)	445
22	3	14	259 (11%)	284
	4	14	391 (17%)	440
	5	15	510 (22%)	465
24	3	15	275 (12%)	281
	4	14	424 (18%)	368
	5	14	555 (24%)	516
26	3	15	296 (13%)	317
	4	15	433 (18%)	415
	5	15	594 (25%)	552
28	3	14	322 (14%)	333
	4	14	479 (20%)	478
	5	14	625 (27%)	633
30	3	13	349 (15%)	319
	4	15	495 (21%)	532
	5	14	653 (28%)	654
32	3	14	344 (15%)	338
	4	14	525 (22%)	508
	5	15	687 (29%)	658

TAB. 3 – *Caractéristiques d'un polynômieur en-ligne exploitant un schéma de Horner en fonction du nombre de chiffres des opérandes et du degré sur un circuit Virtex-E XCV200E-6.*

n	d	τ [ns]	nb CLB	Temps [sec]
10	3	13	140 (6%)	191
	4	15	226 (10%)	277
	5	14	290 (12%)	270
12	3	14	165 (7%)	183
	4	14	246 (10%)	251
	5	14	336 (14%)	320
14	3	14	186 (8%)	201
	4	15	266 (11%)	284
	5	15	368 (16%)	356
16	3	14	197 (8%)	230
	4	13	301 (13%)	290
	5	14	398 (17%)	403
18	3	13	227 (10%)	247
	4	14	327 (14%)	351
	5	14	431 (18%)	467
20	3	14	233 (10%)	248
	4	14	370 (16%)	392
	5	15	467 (20%)	587
22	3	13	262 (11%)	286
	4	14	393 (17%)	378
	5	14	514 (22%)	480
24	3	14	281 (12%)	292
	4	14	411 (17%)	549
	5	14	555 (24%)	670
26	3	15	299 (13%)	440
	4	15	439 (19%)	522
	5	15	598 (25%)	630
28	3	14	321 (14%)	320
	4	15	486 (21%)	427
	5	14	629 (27%)	558
30	3	14	354 (15%)	378
	4	15	499 (21%)	447
	5	15	656 (28%)	647
32	3	14	347 (15%)	365
	4	14	533 (23%)	494
	5	15	688 (29%)	671

TAB. 4 – *Caractéristiques d'un polynômieur en-ligne exploitant un schéma de Horner en fonction du nombre de chiffres des opérandes et du degré sur un circuit Spartan-II XC2S200-6.*

5 Résultats

5.1 Performances du système proposé

Les tableaux 3 et 4 résument les principales caractéristiques du polynômieur de la figure 4 en fonction du nombre de chiffres de l'opérande x et du degré sur des circuits Virtex-E XCV200E-6 et Spartan-II XC2S200-6. Lors de ces expériences, réalisées sur une Sun Ultra-10, avec un processeur cadencé à 440 MHz et 1 Go de RAM, nous avons respectivement utilisé Synplify Pro 7.02 pour la synthèse du code VHDL et la version 4.1.02i des outils de Xilinx pour le placement-routage. Outre la surface et le temps de cycle du polynômieur, nous indiquons le temps nécessaire à la génération du fichier de configuration du FPGA. Remarquons que l'insertion d'un étage de pipeline supplémentaire dans les binômieurs permet une diminution du temps de cycle [4].

Matériel	Schéma 1	Schéma 2	Schéma 3	Schéma 4
Multiplication	1	0	0	1
Addition	1	1	0	2
Décalage	0	1	1	1
$\oplus$	0	0	1	0
Bascule	0	0	1	0
Table [octets]	25 à 28	13 à 14	13 à 14	25 à 28

TAB. 5 – *Ressources matérielles requises par les différents circuits proposés par Vassiliadis* et al..

5.2 Comparaison avec d'autres solutions

Présentons encore brièvement quelques autres approches adoptées pour l'évaluation de fonctions élémentaires et comparons-les à notre méthode.

- La E-méthode est une méthode d'évaluation des fonctions élémentaires basée sur la résolution d'un système linéaire très simple qui calcule en fait une approximation polynomiale. Cette méthode proposée initialement par Ercegovac a été étendue au cas de l'arithmétique en-ligne [8]. Elle donne des opérateurs un peu plus petits que ceux obtenus avec notre polynômieur. Mais elle nécessite des contraintes plus fortes sur les coefficients des polynômes. Ceci conduit à des intervalles plus petits et donc des tables plus grandes. Le débit d'un opérateur de E-méthode est sensiblement le même que celui du polynômieur présenté dans cet article.
- Les méthodes à base de tables multipartites sont un concurrent sérieux aux approximations polynomiales pour de petites précisions. Les tables multipartites n'utilisent que de petites tables, des additions et des fonctions logiques simples comme des `xor` bit à bit. De récentes améliorations [5] permettent d'envisager l'implantation d'opérateurs très rapides et de taille limitée pour une fonction élémentaire jusqu'à une vingtaine de bits. Mais pour implanter plusieurs fonctions à la fois, il faut autant de tables multipartites que de fonctions. L'avantage est donc en la faveur du polynômieur qui est commun à toutes les fonctions. Seules les toutes petites tables qui stockent les coefficients sont propres aux fonctions implantées. À titre de comparaison pour la fonction sinus sur 16 bits, d'après [5], on a sur des circuits Virtex une table multipartite de 166 CLB et de 41 ns de temps de cycle. Pour d'autres fonctions les résultats sont similaires.
- Vassiliadis *et al.* proposent quatre architectures afin d'évaluer diverses fonctions élémentaires intervenant dans les réseaux de neurones artificiels [20]. Les trois premiers systèmes exploitent des approximations linéaires. Un polynôme de degré deux intervient dans le quatrième schéma. Les auteurs suggèrent quelques astuces restreignant les ressources matérielles nécessaires. La discrétisation de certains coefficients à des puissances négatives de deux permet par exemple de remplacer une multiplication par un décalage. Le tableau 5 résume les caractéristiques des différents circuits. En travaillant avec quatre bits de partie entière et dix bits de partie fractionnaire, l'erreur maximale obtenue oscille entre 10^{-3} et 10^{-2} selon la fonction considérée. Les domaines de définition ainsi que la largeur Δ des intervalles dépendent de la fonction considérée. Le sinus et le cosinus sont par exemple évalués sur $]0,\pi]$ avec $\Delta = 0{,}5$.

5.3 Tables des coefficients

Nous présentons ici des résultats de la génération des tables des coefficients pour un exemple réel. La fonction cible est le sinus sur l'intervalle $I = [-1,1]$. Cet intervalle contenant $[-\pi/4,\pi/4]$, on sait donc faire une réduction d'argument additive préalable si nécessaire. La précision souhaitée est environ 15 bits. Nous avons généré les tables de coefficients d'un polynômieur de degré 3 pour différentes tailles de la clé de découpage $m \in \{1,2,3\}$. Les résultats sont présentés table 6. ϵ est la valeur exprimée en nombre de bits justes de $||f(x) - p(x)||_\infty$ sur chaque intervalle I' du découpage présenté à la figure 3.

Pour stocker les tables correspondantes deux solutions sont possibles dans le type de FPGA cible. On peut utiliser soit les blocs de mémoire RAM soit les LUT comme mémoire distribuée. On code les coefficients en signe et valeur absolue car la conversion en *borrow-save* est immédiate. Pour chaque polynôme de degré d, avec des coefficients sur n bits, on a $n(d+1)$ bits à stocker. Le nombre d'intervalles de découpage d'une fonction est $2^{m+1} - 1$ pour une clé sur m chiffres. Le nombre total de bits à stocker pour une fonction est donc $n(d+1)(2^{m+1} - 1)$. Dans notre exemple, on a $d = 3$, $n = 18$ et $m \in \{1,2,3\}$. Le nombre total de bits à

m	I'	$p(x)$	ϵ
1	$[-1,0]$	$\frac{5}{32768} + \frac{16457}{16384}x + \frac{637}{32768}x^2 - \frac{9407}{65536}x^3$	12,6
	$[-\frac{1}{2},\frac{1}{2}]$	$\frac{32763}{32768}x - \frac{5377}{32768}x^3$	15,7
	$[0,1]$	$-\frac{5}{32768} + \frac{16457}{16384}x - \frac{637}{32768}x^2 - \frac{9407}{65536}x^3$	12,6
2	$[-1,-\frac{1}{2}]$	$\frac{435}{65536} + \frac{67857}{65536}x + \frac{2161}{32768}x^2 - \frac{3977}{32768}x^3$	15,9
	$[-\frac{3}{4},-\frac{1}{4}]$	$\frac{51}{65536} + \frac{1031}{1024}x + \frac{659}{32768}x^2 - \frac{2385}{16384}x^3$	16,1
	$[-\frac{1}{2},0]$	$\frac{65555}{65536} + \frac{165}{65536}x^2 - \frac{5267}{32768}x^3$	16,6
	$[-\frac{1}{4},\frac{1}{4}]$	$\frac{65535}{65536}x - \frac{85}{512}x^3$	19,1
	$[0,\frac{1}{2}]$	$\frac{65555}{65536} - \frac{165}{65536}x^2 - \frac{5267}{32768}x^3$	16,6
	$[\frac{1}{4},\frac{3}{4}]$	$-\frac{51}{65536} + \frac{1031}{1024}x - \frac{659}{32768}x^2 - \frac{2385}{16384}x^3$	16,1
	$[\frac{1}{2},1]$	$-\frac{435}{65536} + \frac{67857}{65536}x - \frac{2161}{32768}x^2 - \frac{3977}{32768}x^3$	15,9
3	$[-1,-\frac{3}{4}]$	$\frac{501}{32768} + \frac{34951}{32768}x + \frac{845}{8192}x^2 - \frac{6993}{65536}x^3$	17,2
	$[-\frac{7}{8},-\frac{5}{8}]$	$\frac{59}{8192} + \frac{67941}{65536}x + \frac{4345}{65536}x^2 - \frac{7983}{65536}x^3$	19,0
	$[-\frac{3}{4},-\frac{1}{2}]$	$\frac{3}{1024} + \frac{33357}{32768}x + \frac{1279}{32768}x^2 - \frac{553}{4096}x^3$	18,9
	$[-\frac{5}{8},-\frac{3}{8}]$	$\frac{31}{32768} + \frac{66021}{65536}x + \frac{83}{4096}x^2 - \frac{9575}{65536}x^3$	18,3
	$[-\frac{1}{2},-\frac{1}{4}]$	$\frac{7}{32768} + \frac{8211}{8192}x + \frac{283}{32768}x^2 - \frac{1269}{8192}x^3$	18,6
	$[-\frac{3}{8},-\frac{1}{8}]$	$\frac{1}{32768} + \frac{65565}{65536}x + \frac{21}{8192}x^2 - \frac{5285}{32768}x^3$	17,6
	$[-\frac{1}{4},0]$	$\frac{65537}{65536}x + \frac{5}{16384}x^2 - \frac{10825}{65536}x^3$	22,1
	$[-\frac{1}{8},\frac{1}{8}]$	$x - \frac{10913}{65536}x^3$	23,9
	$[0,\frac{1}{4}]$	$\frac{65537}{65536}x - \frac{5}{16384}x^2 - \frac{10825}{65536}x^3$	22,1
	$[\frac{1}{8},\frac{3}{8}]$	$-\frac{1}{32768} + \frac{65565}{65536}x - \frac{21}{8192}x^2 - \frac{5285}{32768}x^3$	17,6
	$[\frac{1}{4},\frac{1}{2}]$	$-\frac{7}{32768} + \frac{8211}{8192}x - \frac{283}{32768}x^2 - \frac{1269}{8192}x^3$	18,6
	$[\frac{3}{8},\frac{5}{8}]$	$-\frac{31}{32768} + \frac{66021}{65536}x - \frac{83}{4096}x^2 - \frac{9575}{65536}x^3$	18,3
	$[\frac{1}{2},\frac{3}{4}]$	$-\frac{3}{1024} + \frac{33357}{32768}x - \frac{1279}{32768}x^2 - \frac{553}{4096}x^3$	18,9
	$[\frac{5}{8},\frac{7}{8}]$	$-\frac{59}{8192} + \frac{67941}{65536}x - \frac{4345}{65536}x^2 - \frac{7983}{65536}x^3$	19,0
	$[\frac{3}{4},1]$	$-\frac{501}{32768} + \frac{34951}{32768}x - \frac{845}{8192}x^2 - \frac{6993}{65536}x^3$	17,2

TAB. 6 – *Contenu des tables pour la fonction sinus pour différents découpages.*

stocker est donc de 216, 504 et 1080 pour $m = 1$, 2 et 3 respectivement. Nous avons fait une synthèse dans les mêmes conditions que précédemment pour chaque découpage, on obtient 4, 7 et 12 CLB pour $m = 1$, 2 et 3 respectivement. Les temps d'adressages des tables sont toujours bien plus faibles que le temps de cycle des opérateurs générés. Toutefois, dans le cas de l'utilisation des blocs de mémoire, le placement et le routage jouent un rôle important sur la vitesse du circuit. Étant donnée la taille des tables générées, il est effectivement envisageable d'évaluer plusieurs fonctions avec le même opérateur.

5.4 Qualité numérique des résultats

Dans la section 3, nous avons indiqué que la précision retournée par notre programme Maple est seulement basée sur l'erreur de méthode et la mise au format final des coefficients du polynôme. En particulier, les erreurs d'arrondis des calculs intermédiaires de l'évaluation ne sont pas prises en compte. Nous illustrons ici cette différence de précision à l'aide d'un exemple.

La fonction choisie est $\sin$ sur l'intervalle $[0,\pi/8]$. La précision cible est 8 bits. Nous prenons une toute petite précision car nous allons devoir effectuer des simulations exhaustives de l'opérateur généré. Le programme Maple retourne le polynôme :

$$p(x) = \frac{259}{256}x - \frac{3}{32}x^2$$

L'erreur de cette approximation, c'est-à-dire $||f(x) - p(x)||_\infty$ sur I, est de $4{,}45.10^{-4}$ soit 11,1 bits de précision. Avec les coefficients $p_0 = 0$, $p_1 = 259/256$ et $p_2 = -3/32$, on aurait donc un peu plus de 11 bits justes si l'évaluation était effectuée exactement. Nous avons généré l'opérateur VHDL correspondant et puis nous l'avons simulé exhaustivement pour mesurer la précision réelle. C'est-à-dire que pour chaque valeur possible de x, on simule l'architecture complète et on enregistre sa sortie.

La figure 6 représente la courbe des résultats de l'opérateur et celle de l'erreur totale. La courbe du haut est simplement la courbe des couples $(x,q(x))$ où $q(x)$ est le résultat du polynôme $p(x)$ calculé effectivement sur le FPGA. La courbe de dessous donne la courbe des points $(x,\epsilon_T(x))$ où $\epsilon_T(x) = q(x) - \sin(x)$. L'erreur $\epsilon_T(x)$ est donc bien l'erreur totale, elle intègre à la fois l'erreur de méthode et l'erreur de calcul. La plus grande valeur de $\epsilon_T(x)$ mesurée est de $4{,}04.10^{-3}$ soit 7,95 bits de précision pour $x = \pi/8$.

Du fait de la perte de précision lors de l'évaluation, il faut donc « surdimensionner » l'approximation. Dans les opérateurs en-ligne, on tronque le résultat après le dernier chiffre souhaité. On peut donc perdre jusqu'à un bit de précision par opération, par exemple, lorsque le dernier chiffre souhaité est un 0 alors que tous les chiffres suivants sont des 1 (ou tous des -1). Lors de l'évaluation d'un polynôme de degré d, on a d additions et d multiplications. Soit une perte totale bornée par $2d$ bits. En pratique, cette borne semble être loin d'être atteinte. Nous n'avons pas trouvé dans la littérature de travaux sur ce sujet; c'est un point sur lequel nous comptons travailler dans l'avenir.

Un dernier problème porte sur la conservation de propriétés plus que sur la précision. Dans notre exemple ci-dessus, on constate que le coefficient p_0 est nul, ce qui assure que $\sin(0) = 0$ lors du calcul. Mais il n'en est pas toujours ainsi. Par exemple, si on cherche à approcher la fonction e^x sur l'intervalle $[-1/2,1/2]$, on est en droit de souhaiter avoir le résultat $e^0 = 1$. Mais voici ce que donne notre programme Maple pour un polynôme de degré 3 avec 18 chiffres dont 16 fractionnaires :

$$p(x) = \frac{8189}{8192} + \frac{4095}{4096}x + \frac{2091}{4096}x^2 + \frac{1391}{8192}x^3$$

Ceci n'entraîne toutefois pas nécessairement un problème de précision du calcul. Comme on doit calculer sur une précision interne plus grande que la précision cible, cette petite erreur en $x = 0$ peut très bien être inférieur à l'erreur maximale souhaitée. Une solution que nous comptons étudier dans l'avenir est de forcer la valeur de certains coefficients, dans notre exemple on forcerait $p_0 = 1$, tout en maintenant l'erreur totale en dessous d'une valeur donnée.

6 Conclusion

L'évaluation des fonctions élémentaires a été abordée dans le cadre particulier de l'arithmétique en-ligne matérielle. L'utilisation d'une approximation polynomiale permet la réutilisation d'une architecture de calcul

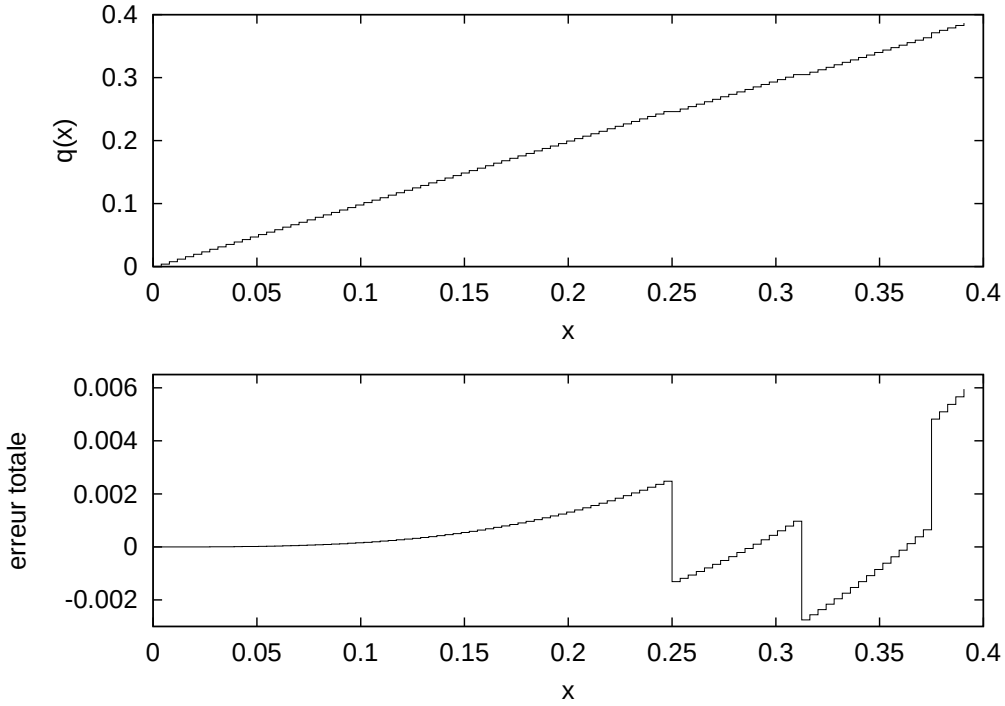

FIG. 6 – *Résultat réel et son erreur total d'un polynomieur de degré 2 pour l'évaluation de* $\sin$ *sur* $[0,\pi/8]$.

pour différentes fonctions. En effet, la majeure partie du polynômieur en-ligne réalisé est générique, seules les petites tables stockant les coefficients sont propres aux fonctions évaluées. Nous avons réalisé des générateurs automatiques à la fois pour la détermination des coefficients des polynômes et pour une description VHDL synthétisable de l'architecture complète, pour varier la taille des opérandes. Le polynômieur ainsi réalisé permet d'évaluer rapidement quelques fonctions élémentaires avec une surface de circuit modérée et un bon débit.

Remerciements

Nous tenons à remercier chaleureusement le Ministère de la Recherche Français pour son support avec l'« ACI jeunes chercheurs », le Fonds National Suisse de la Recherche Scientifique et la société Xilinx pour son don de circuits FPGA via le *Xilinx University Program*.

Références

[1] A. Aggoun, M.K. Ibrahim, and A. Ashur. Bit-level pipelined digit-serial array processors. *IEEE Transactions on Circuits and Systems-II*, 45(7):857–868, July 1998.

[2] A. Avizienis. Signed-Digit Number Representations for Fast Parallel Arithmetic. *IRE Transactions on Electronic Computers*, 10:389–400, 1961.

[3] J.C. Bajard, J. Duprat, S. Kla, and J.M. Muller. Some operators for on-line radix-2 computations. *Journal of Parallel and Distributed Computing*, 22:336–345, 1994.

[4] J.-L. Beuchat. *Étude et conception d'opérateurs arithmétiques optimisés pour circuits programmables*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2001. Thèse No 2426.

[5] F. de Dinechin and A. Tisserand. Some improvements on mutlipartite tables methods. In *15th International Symposium on Computer Arithmetic ARITH15*, Vail, Colorado, June 2001. IEEE.

[6] M. Dimmler, A. Tisserand, U. Holmberg, and R. Longchamp. On-line arithmetic for real-time control of microsystems. *IEEE/ASME Transactions on Mechatronics*, 4(2):213–217, June 1999.

[7] M. D. Ercegovac and T. Lang. On-Line Arithmetic: A Design Methodology and Applications in Digital Signal Processing. In *IEEE Acoustics, Speech, and Signal Processing Society Workshop on VLSI Signal Processing*, pages 252–263, November 1988.

[8] M.D. Ercegovac, J.M. Muller, and A. Tisserand. FPGA implementation of polynomial evaluation algorithm. In SPIE, editor, *Field Programmable Gate Arrays for Fast Board Development and Reconfigurable Computing*, volume 2607, pages 177–188. SPIE, October 1995. Philadelphia, Pennsylvania.

[9] B. Girau and A. Tisserand. On-line Arithmetic-Based Reprogrammable Hardware Implementation of Multilayer Perceptron Back-Propagation. In *Proceedings of MicroNeuro '96*, pages 168–175, 1996.

[10] R. Hartley and P. Corbett. Digit-serial processing techniques. *IEEE Transactions on Circuits and Systems*, 37(6):707–719, June 1990.

[11] S. Kla. *Calcul parallèle et en-ligne des fonctions arithmétiques*. PhD thesis, Ecole Normale Supérieure de Lyon – Laboratoire de l'Informatique du Parallélisme, February 1993.

[12] E. Mosanya. *A Reconfigurable Processor for Biomolecular Sequence Processing*. PhD thesis, École Polytechnique Fédérale de Lausanne, 1998. Thèse No 1910.

[13] J.-M. Muller. *Elementary Functions: Algorithms and Implementation*. Birkhauser, Boston, 1997.

[14] A.M. Nielsen and J.M. Muller. On-line algorithms for computing exponentials and logarithms. In *Europar'96*, Lyon, France, August 1996.

[15] K.K. Parhi. A systematic approach for design of digit-serial signal processing architectures. *IEEE Transactions on Circuits and Systems*, 38(4):358–375, April 1991.

[16] S. Rajagopal and J. R. Cavallaro. On-line Arithmetic for Detection in Digital Communication Receivers. In *Proceedings of the 15th IEEE International Symposium on Computer Arithmetic*, pages 257–265, 2001.

[17] E. Remez. Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. *C.R. Académie des Sciences*, 198, 1934. Paris.

[18] A. Skaf, J.-M. Muller, and A. Guyot. On-Line Hardware Implementation for Complex Exponential and Logarithm. In *Proceedings of the 20th European Solid-State Circuits Conference*, September 1994.

[19] K. S. Trivedi and M. D. Ercegovac. On-line Algorithms for Division and Multiplication. *IEEE Transactions on Computers*, C–26(7), July 1977.

[20] S. Vassiliadis, M. Zhang, and J. G. Delgado-Frias. Elementary Function Generators for Neural-Network Emulators. *IEEE Transactions on Neural Networks*, 11(6):1438–1449, November 2000.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399