Small FPGA polynomial approximations with 3-bit coefficients and low-precision estimations of the powers of x

Romain Michard, Arnaud Tisserand, Nicolas Veyrat-Charvillon

▶ **To cite this version:**

Romain Michard, Arnaud Tisserand, Nicolas Veyrat-Charvillon. Small FPGA polynomial approximations with 3-bit coefficients and low-precision estimations of the powers of x. [Research Report] RR-5503, LIP RR-2005-8, INRIA, LIP. 2005, pp.13. inria-00070504

HAL Id: inria-00070504

https://inria.hal.science/inria-00070504v1

Submitted on 19 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Small FPGA polynomial approximations with 3-bit coefficients and low-precision estimations of the powers of x

Romain Michard, Arnaud Tisserand ant Nicolas Veyrat-Charvillon

N° 5503

February 2005

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--5503--FR+ENG

INRIA
RHÔNE-ALPES

Small FPGA polynomial approximations with 3-bit coefficients and low-precision estimations of the powers of x

Romain Michard, Arnaud Tisserand ant Nicolas Veyrat-Charvillon

Thème SYM — Systèmes symboliques
Projet Arénaire

Rapport de recherche n° 5503 — February 2005 — 13 pages

Abstract: This paper presents small FPGA implementations of low precision polynomial approximations of functions without multipliers. Our method uses degree-2 or degree-3 polynomial approximations with at most 3-bit coefficients and low precision estimations of the powers of x. Here, we denote by 3-bit coefficients values with at most 3 non-zero and possibly non-contiguous signed bits (e.g. $1.001000\overline{1}$). This leads to very small operators by replacing the costly multipliers by a small number of additions. Our method provides approximations with very low average error and is suitable for signal processing applications.

Key-words: computer arithmetic, hardware arithmetic operator, polynomial evaluation, digital signal application.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Petites approximations polynomiales pour FPGA avec des coefficients à 3 bits et estimation à basse précision des puissances de x

Résumé : Ce papier présente une implantation sur FPGA d'approximations polynomiales de fonctions à faible précision sans multiplieur. Notre méthode utilise des approximations polynomiales de degré 2 ou 3 avec des coefficients sur au plus 3 bits et des estimations à faible précision des puissances de x. On entend par coefficients à 3 bits des représentations binaires avec au plus 3 bits signés et potentiellement non contigus (par exemple $1.001000\overline{1}$). Ceci conduit à de très petits opérateurs en remplaçant les coûteux multiplieurs par un petit nombre d'additions. Notre méthode fournit des approximations avec une très faible erreur moyenne, elle est donc intéressante pour des applications de traitement du signal.

Mots-clés : arithmétique des ordinateurs, opérateur arithmétique matériel, évaluation de polynôme, application au traitement du signal.

1 Introduction

In digital systems, polynomial approximations are widely used. The elementary functions (e.g. sine, cosine, logarithm, exponential), for instance, are often evaluated using polynomials [7]. Algebraic functions, such as square root or reciprocal square root can be efficiently approximated using polynomials. Low-degree polynomials are often used for evaluating reciprocals in digital signal processing applications such as frequency demodulation.

The size of the multipliers is often a problem when implementing function approximations in hardware. Several solutions have been investigated to limit their size. Methods based on tables and small multiplications are often used [13, 8, 4, 2]. For small precision, some methods, such as the multipartite tables, have been introduced to avoid the use of multipliers [12, 10, 1]. The partial product arrays (PPAs) method [11, 5] uses converging series where all the operations have been developed at the bit level and where the low-weight terms are discarded.

In this work we focus on polynomial approximations with at most 3-bit coefficients and low-precision estimations of the powers of x. We deal with degree-2 or degree-3 polynomial approximations: $P(x) = p_0 + p_1 x + p_2 x^2 [+p_3 x^3]$. The coefficients p_1, p_2 and p_3 are represented using at most 3 non-zero signed bits in order to replace the multipliers by a small number of additions. The coefficient p_0 is kept as large as possible since it is an additive term. In order to further decrease the size of the operators, we use low-precision estimations of the powers of x (i.e. x^2 and x^3). The proposed method leads to approximations with a maximum error limited to a few LSBs, but with very low average error. The obtained average error is close to the error of the minimax polynomial. This makes our method an attractive solution for some applications in digital signal processing.

This paper is organized as follows. The notations and background on polynomial approximations are presented in Section 2. Our contribution is presented in Section 3. Section 4 presents the FPGA implementation of our operators. We compare our results with other methods in Section 5. We conclude and give future prospects in Section 6.

2 Notations and Minimax Polynomial Approximations

In this work, we deal with the evaluation of a function f with inputs and outputs in fixed-point format. The argument x is in the domain $[a, b[$ and the result $f(x)$ is in the range $[a', b'[$ (or $]a', b']$). Our work can be straightforwardly extended to other forms of intervals (e.g. $[a, b]$). The integer d denotes the degree of the polynomials. The argument x is a w_I-bit number and the output $f(x)$ is a w_O-bit number. The notation $()_2$ denotes the binary representation of a value. For instance the value 3.125 is represented in binary by $(11.001)_2$. The quantified coefficients of the polynomial will be represented in the *borrow-save* format [3]. Bits with a negative weight are denoted by $\overline{1}$.

The input argument x is considered as exact, we will ignore the question of input discretisation. The *approximation error* measures the distance between the mathematical function f and the approximated function used to evaluate it. The *rounding error* due to the dis-

crete nature of the final and intermediate values adds up to the approximation error. In order to limit the rounding error, we introduce g additional *guards bits* for the intermediate computations (i.e. the intermediate computations are done on words of $w_O + g$ bits).

In order to measure the theoretical approximation error ϵ_{th} due to the use of the polynomial P to evaluate the function f on $[a, b]$, we use the distance:

$$\epsilon_{th} = ||f - P||_\infty = \max_{a \le x \le b} |f(x) - P(x)|,$$

an estimation of this distance is obtained using the Maple `infnorm` function.

For a given argument x it is possible to evaluate the effective total error $\epsilon = f(x) - \texttt{output}(P(x))$ which includes all kinds of error. Here, $\texttt{output}(P(x))$ is the result of the evaluation of $P(x)$ by the circuit using finite precision computations. As the proposed method is limited to low-precision approximations (up to 16 bits), the *average* total error ϵ_{avg} and its *standard deviation* σ can be computed. Indeed, for all possible values of x, the effective result $\texttt{output}(P(x))$ is evaluated and compared to the theoretical value $f(x)$. The effective *maximum* error ϵ_{max} can also be computed.

The polynomial approximations used in the following are based on the *minimax* polynomial approximation as a starting point. The degree-d minimax polynomial approximation to f on $[a, b]$ is the polynomial P^* that satisfies:

$$||f - P^*||_\infty = \min_{P \in \mathcal{P}_d} ||f - P||_\infty,$$

where $\mathcal{P}_d$ is the set of polynomials with real coefficients and degree at most d. Minimax approximations can be computed thanks to an algorithm due to Remez [9] (available as the `minimax` function in Maple).

Example 1 *Degree-3 minimax approximation of the sine function on* $[0, \pi/4]$ *(results from Maple and truncated to* 10 *decimals),* $\epsilon_{th} = 0.0000474552$ *(i.e.* 14.3 *bits of accuracy):*

$$P(x) = -0.0000474552 + 1.0017332478\,x - 0.0095826177\,x^2 - 0.1522099691\,x^3.$$

In the following, an error ϵ is expressed using two equivalent values. Its actual value ϵ, and its value expressed in number of correct bits (reported between parenthesis). For instance, the error $\epsilon_{th} = 0.0023098047$ is equivalent to an accuracy of 8.7 correct bits.

3 The 3-bit coefficients and estimations of powers of x method

The proposed method is based on the following steps:

Step 1 determination of the minimax polynomial approximation P_{th} (done using Maple);

Step 2 quantification of the coefficients of P_{th} to 3 non-zero bits, this gives polynomial P_q;

Step 3 estimation of the powers of x in polynomial P_q;

Step 4 fine tuning of the coefficients, this gives the polynomial P_t.

3.1 3-bit Quantification of the Coefficients

In order to replace the large reduction tree of the multipliers by a small number of additions, the coefficients (p_1, p_2 and p_3 if any) of polynomial P_{th} are quantified to values with at most 3 non-zero bits. The constant coefficient p_0 is not quantified. It is kept as large as possible since it is an additive term.

The quantification of coefficient p_i with NZ non-zero bits (here $NZ \leq 3$) and a relative accuracy of 2^{-k} (the span of the NZ bits is at most k-bit wide) is obtained using an iterative algorithm. At each iteration, the power of 2 the nearest to p_i is determined and subtracted to p_i. This iteration is applied to the remainder until NZ non-zero bits are used or an accuracy less than 2^{-k} is reached (when the remainder is zero or the difference of ranks between the most significant and the least significant non-zero bits is greater than k).

The result of the quantification step is one of the possible representations of p_i with at most NZ non-zero bits and an accuracy less than or equal to 2^{-k}. The returned quantified coefficient also has the smallest possible span (the difference between the ranks of the most significant and least significant non-zero bits). For instance, the algorithm favors the case $(11.01)_2$ rather the case $(10\overline{1}.01)_2$ for the value 3.25.

Depending on the target, some other equivalent representations may be preferable. For instance, if ASIC implementations are targeted, it may be more efficient to favor the representation with the smallest number of negative bits (a subtraction may be slightly larger than an addition). In this case, the quantification of the value 1.375 with the representation $(1.011)_2$ should be prefered to the equivalent representation $(1.10\overline{1})_2$.

Example 2 *Quantification of the value* $v = 1.0017332478$ *with* 3 *non-zero bits for several spans (the quantified value is denoted* v_q*):*

- *for* $k = 8$ *bits,* $v_q = 2^0 = (1.0000000)_2$, $v - v_q = 0.0017332478$ *(i.e.* 9.1 *bits of accuracy);*
- *for* $k = 10$ *bits,* $v_q = 2^0 + 2^{-9} = (1.000000001)_2$, $v - v_q = -0.0002198772$ *(i.e.* 12.1 *bits of accuracy);*
- *for* $k = 13$ *bits,* $v_q = 2^0 + 2^{-9} - 2^{-12} = (1.00000000100\overline{1})_2$, $v - v_q = 0.0000242634$ *(i.e.* 15.3 *bits of accuracy).*

Figure 1 presents the quantification error for all the 10-bit values of x in $[0, 1[$ quantified to 3 non-zero bits with a maximum span of $k = 8$ bits. The average error is 0.0000534057 (i.e. 14.1 bits of accuracy), its standard deviation is 0.0031827562. The maximum error is 0.015625 (i.e. 4 LSBs).

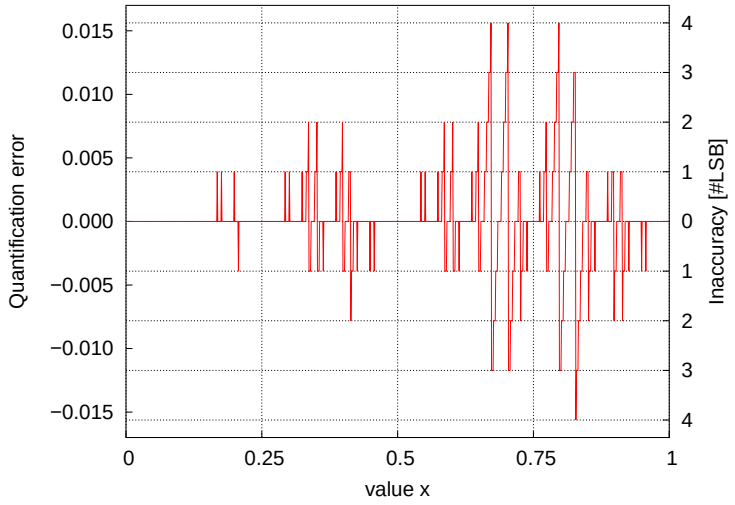

Figure 1: Quantification error for 10-bit values in $[0, 1[$ with $NZ = 3$ and $k = 8$.

3.2 Low-Precision Estimations of the Powers of x

Once the coefficients of the polynomial have been quantified to 3-bit values, some multiplications remain during the evaluation of $P(x)$. Indeed the square x^2 and the cube x^3 of the argument x have to be computed. In this work we also try to replace these "multiplications" by a small number of additions. For that, we replace the values of x^2 and x^3 by estimations of these values.

First of all, we apply the standard simplifications for the computations of the partial products of x^2 and x^3. Those simplifications can be found in [3, 6].

The estimation is done by taking into account only the c first columns of the partial products. The number of columns c is a parameter in the exploration space.

Table 1 presents the impact of the number of columns c in the estimation of x^2 on the evaluation of $p_2 \times x^2$ with $p_2 = 0.1882871881$, $w_I = k = 8$ bits and $NZ = 3$. For each value of c, several values are reported: the number of partial products $\#_{pp}$, the average error ϵ_{avg} and its standard deviation σ and the maximum error ϵ_{max}. Those error characteristics are evaluated for the 2^8 possible values of x in $[0, \pi/4[$. The number of partial products when all the columns are used in x^2 is 36 and not $64 = 8^2$ due to the simplifications from [3, 6].

The values of the average error reported in Table 1 show that the loss of accuracy in the computation of $p_2 \times x^2$ may be large when c is small. One can conclude that it is not a good idea to estimate the powers of x. This is false! Table 2 presents the same study for the *complete* computation of the sine function on $[0, \pi/4[$. The polynomial used with the

c	$\#_{pp}$	ϵ_{avg}	σ	ϵ_{max}
3	6	0.16×10^{-1} (5.9)	0.12×10^{-1}	0.47×10^{-1} (4.4)
4	8	0.78×10^{-2} (6.9)	0.58×10^{-2}	0.23×10^{-1} (5.4)
5	12	0.59×10^{-2} (7.4)	0.42×10^{-2}	0.18×10^{-1} (5.8)
6	16	0.28×10^{-2} (8.4)	0.19×10^{-2}	0.88×10^{-2} (6.8)
7	20	0.18×10^{-2} (9.0)	0.11×10^{-2}	0.52×10^{-2} (7.5)
all	36	0.13×10^{-2} (9.5)	0.67×10^{-3}	0.32×10^{-2} (8.3)

Table 1: Size and accuracy of the evaluation of $p_2 \times x^2$ for several values of c.

parameters $w_I = w_O = k = 8$, $g = 2$ and $NZ = 3$ is:

$$p(x) = -(0.000000001)_2 + (1.000100\overline{1})_2 \times x - (0.0011000001)_2 \times x^2.$$

For this polynomial several values of c have been tested for the estimation of x^2. The error characteristics are evaluated for the 2^8 possible values of x in $[0, \pi/4[$. Table 2 clearly shows that a rough estimation of x^2 is sufficient to provide a correct average error and maximum error. For this example, approximations with only $c = 5$ or 6 columns (for x^2) have an accuracy equivalent to the solution with the complete, and costly, computation of x^2.

c	ϵ_{avg}	σ	ϵ_{max}
3	0.62×10^{-2} (7.3)	0.49×10^{-2}	0.23×10^{-1} (5.4)
4	0.44×10^{-2} (7.8)	0.35×10^{-2}	0.15×10^{-1} (6.0)
5	0.23×10^{-2} (8.7)	0.16×10^{-2}	0.75×10^{-2} (7.0)
6	0.22×10^{-2} (8.8)	0.15×10^{-2}	0.75×10^{-2} (7.0)
7	0.21×10^{-2} (8.8)	0.15×10^{-2}	0.75×10^{-2} (7.0)
all	0.21×10^{-2} (8.8)	0.15×10^{-2}	0.75×10^{-2} (7.0)

Table 2: Accuracy of the evaluation of $sin(x)$ on $[0, \pi/4[$ for several values of c.

3.3 Fine Tuning of the p_i's

Due to the estimation of the powers of x, the overall accuracy can be slightly improved by modifying the value of the quantified coefficients. Indeed, using only the c most significant columns in the partial products of x^i leads to underestimate its actual value. In order to compensate this underestimation, one can try to modify the coefficient p_i.

The fine tuning algorithm is quite simple. For each monomial $p_i x^i$, it determines its average error ϵ_i. If ϵ_i is less than zero, 1 is added to the LSB of the quantified coefficient p_i (-1, if $\epsilon_i > 0$). This step is repeated while the accuracy of the result is improved. The result of the fine tuning step depends on the actual value of c.

Example 3 *Fine tuning of the sine function on* $[0, \pi/4[$ *with* $w_I = w_O = 12$, $NZ = 3$ *and* $g = 2$. *The estimations of* x^2 *is done with* $c = 4$ *and* $c = 8$ *for* x^3. *Before the fine tuning,* $\epsilon_{avg} = 0.91 \times 10^{-3}\,(9.9)$ *with the coefficients:*

$$p_1 = 2^0 + 2^{-9} - 2^{-12},\ p_2 = -2^{-7} - 2^{-9} + 2^{-13},\ p_3 = -2^{-3} - 2^{-5} + 2^{-8}.$$

Fine tuning on the individual coefficients:

- p_1*: no modification;*
- p_2*:* $-2^{-7} - 2^{-9} + 2^{-13}$ *modified to* $-2^{-7} - 2^{-9}$ *(*ϵ_2 *:* $-0.27 \times 10^{-3} \longrightarrow 0.26 \times 10^{-3}$*);*
- p_3*:* $-2^{-3} - 2^{-5} + 2^{-8}$ *modified to* $-2^{-3} - 2^{-5}$ *(*ϵ_3 *:* $-0.84 \times 10^{-3} \longrightarrow 0.56 \times 10^{-3}$*).*

After the fine tuning, $\epsilon_{avg} = 0.72 \times 10^{-3}\,(10.4)$ *and two additions have been suppressed.*

3.4 A Complete Example

Let us approximate the sine function on $[0, \pi/4[$ with 8-bit input and output ($w_I = w_O = 8$) with a degree-2 polynomial. The corresponding minimax polynomial is:

$$P_{th}(x) = -0.0023098047 + 1.0540785973\,x - 0.1882871881\,x^2.$$

The quantification step is done with the parameters $k = 8$, $NZ = 3$ and $g = 2$. The result of the quantification step is the polynomial:

$$P_q(x) = -(0.000000001)_2 + (1.0001001\overline{1})_2\,x - (0.0011000001)_2\,x^2.$$

The next step is the estimation of x^2. Here we only use $c = 3$ columns. The value of the coefficients of $P_q(x)$ do not change during this step.

The last step is the fine tuning of the coefficient p_2. In this case, due to the hazard, the number of non-zero bits of the new coefficient is decreased. The result is the polynomial:

$$P_t(x) = -(0.000000001)_2 + (1.0001001\overline{1})_2\,x - (0.0100\overline{1})_2\,x^2.$$

The accuracy measured for each step is reported in Table 3. The results show that the 3-bit quantification is not the main source of error. In this case, the very rough estimation of x^2 (only 3 columns) leads to an average accuracy of 7 bits. After the fine tuning step, the average error is slightly improved. These results show that even with very "cheap" estimation of the powers of x, reasonable average accuracy can be achieved using our method. Figure 2 presents an illustration of the overall computation for this example.

3.5 Some Accuracy Results

Table 4 summarizes the accuracy results for the sine function. For degree-3 approximations, the numbers of columns are given in the order x^2, x^3. These results show that correct approximations can be achieved using rough estimations of x^2 and x^3. Table 5 presents accuracy results for other common functions. For function 2^x with a degree-3 polynomial, using only $NZ = 3$ for coefficient p_1 leads to a very small accuracy. This can be improved using $NZ = 4$ for this specific coefficient (line $d = 3^\star$ in Table 5).

Step	ϵ_{avg}	σ	ϵ_{max}	Cost
Minimax P_{th}	$\epsilon_{th} = 0.23 \times 10^{-2}$ (8.7)			3 mult. + 2 add.
Quantification	0.16×10^{-2} (9.3)	0.12×10^{-2}	0.53×10^{-2} (7.5)	1 mult. + 2 add.
x^2 estimation	0.69×10^{-2} (7.1)	0.52×10^{-2}	0.23×10^{-1} (5.4)	7 add.
Fine tuning	0.41×10^{-2} (7.5)	0.41×10^{-2}	0.18×10^{-1} (5.8)	6 add.

Table 3: Accuracy evolution for the different steps for $sin(x)$ on $[0, \pi/4[$.

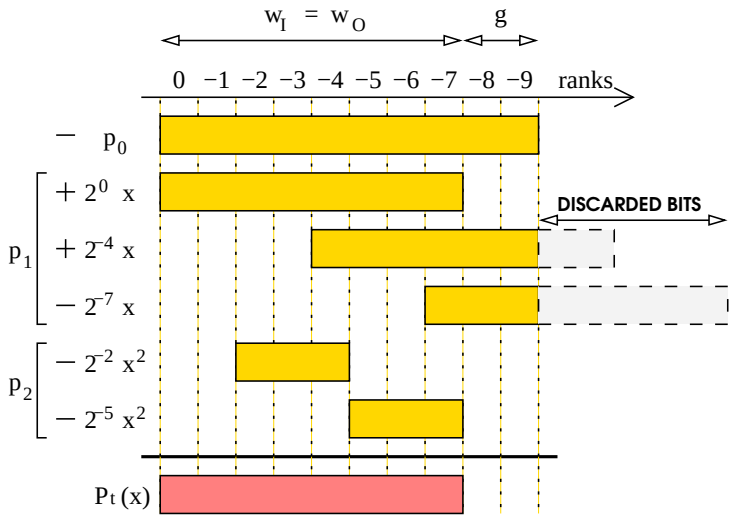

Figure 2: Illustration of the computation of P_t for $sin(x)$ on $[0, \pi/4[$.

4 FPGA Implementations

All the presented results have been obtained for Xilinx Virtex-E XCV400-e FPGA with the ISE 5.2.03i environment. Standard effort has been used both for synthesis and for place-and-route (P&R) steps. The reported results are post-P&R values.

Table 6 presents the results of the implementation of different versions of multiplier-based polynomiers for several size and degree parameters. Those versions are:

- generic : full-width non-constant coefficients, multipliers for $p_i \times x^i$ and optimized multipliers for x^i;
- constant : full-width but constant coefficients (for sine function on $[0, \pi/4[$), constant multipliers for $p_i \times x^i$ and optimized multipliers for x^i;

Target	c	ϵ_{avg}	σ	ϵ_{max}
$w_I = 8$	3	0.57×10^{-2} (7.4)	0.41×10^{-2}	0.19×10^{-1} (5.7)
$d = 2$	4	0.43×10^{-2} (7.8)	0.30×10^{-2}	0.14×10^{-1} (6.1)
	5	0.24×10^{-2} (8.7)	0.17×10^{-2}	0.09×10^{-1} (6.7)
	6	0.18×10^{-2} (9.1)	0.12×10^{-2}	0.05×10^{-1} (7.6)
	all	0.16×10^{-2} (9.3)	0.11×10^{-2}	0.05×10^{-2} (7.6)
$w_I = 12$	1,7	0.87×10^{-3} (10.1)	0.86×10^{-3}	0.51×10^{-2} (7.6)
$d = 3$	2,8	0.61×10^{-3} (10.6)	0.57×10^{-3}	0.31×10^{-2} (8.3)
	3,10	0.44×10^{-3} (11.1)	0.38×10^{-3}	0.24×10^{-2} (8.7)
	4,11	0.38×10^{-3} (11.3)	0.30×10^{-3}	0.16×10^{-2} (9.2)
	5,12	0.17×10^{-3} (12.5)	0.13×10^{-3}	0.08×10^{-2} (10.2)
	all	0.08×10^{-3} (13.5)	0.06×10^{-3}	0.03×10^{-2} (11.7)

Table 4: Accuracy results for $sin(x)$ on $[0, \pi/4[$, $w_I = w_O = k$, $NZ = 3$,$g = 2$.

f	d	w_I	c	ϵ_{avg}	σ	ϵ_{max}
$1/x$	2	8	5	0.39×10^{-2} (8.0)	0.31×10^{-2}	0.24×10^{-1} (5.4)
$[1, 2[$	3	12	9, 9	0.10×10^{-2} (9.9)	0.78×10^{-3}	0.36×10^{-2} (8.1)
2^x	2	8	6	0.37×10^{-2} (8.0)	0.27×10^{-2}	0.11×10^{-1} (6.5)
$[0, 1[$	3	12	6, 6	0.68×10^{-2} (7.2)	$0. \times 36^{-2}$	$0. \times 21^{-1}$ (5.5)
	$3^\star$	12	6, 6	0.19×10^{-2} (9.0)	$0. \times 15^{-2}$	$0. \times 08^{-1}$ (6.9)
$\sqrt{x}$	2	8	5	0.15×10^{-2} (9.3)	0.11×10^{-2}	0.52×10^{-2} (7.5)
$[1, 2[$	3	12	7, 7	0.21×10^{-3} (12.2)	0.16×10^{-3}	0.98×10^{-3} (9.9)
$1/\sqrt{x}$	2	8	5	0.32×10^{-2} (8.3)	0.22×10^{-2}	0.11×10^{-1} (6.5)
$[1, 2[$	3	12	7, 7	0.89×10^{-3} (10.1)	0.77×10^{-3}	0.40×10^{-2} (7.9)

Table 5: Accuracy results for other functions.

- quantified : quantified coefficients (for sine function on $[0, \pi/4[$, $k = w_I$ and $NZ = 3$), additions for $p_i \times x^i$ and optimized multipliers for x^i;

Table 7 presents the implementation results for polynomial approximations of sine function on $[0, \pi/4[$ with coefficients quantified to 3-bit values and estimation of x^2 or x^3. These fully optimized results show significant improvements both for speed and area. Table 8 presents results for other functions and parameters.

Version	d	2			3		
	$w_I = w_O$	8	12	16	8	12	16
Generic	Area [# slices]	115	235	411	324	895	1886
	Period [ns]	24.8	35.9	37.9	38.6	54.6	50.4
Constant	Area [# slices]	43	107	192	204	697	1570
	Period [ns]	15.2	27.9	29.9	33.9	52.3	59.0
Quantified	Area [# slices]	43	85	136	202	651	1475
	Period [ns]	14.3	21.3	22.7	26.8	35.5	38.5

Table 6: FPGA implementation results for generic polynomiers (using multipliers) for $\sin(x)$ on $[0, \pi/4[$.

Version	d	2			3		
	$w_I = w_O$	8	12	16	8	12	16
$c = 6$	Area [# slices]	34	50	61	56	78	99
	Period [ns]	15.1	18.7	19.5	21.5	23.7	22.9
$c = 4$	Area [# slices]	28	46	57	38	61	82
	Period [ns]	14.8	18.6	18.9	18.3	20.1	22.1
$c = 3$	Area [# slices]	27	45	56	28	50	71
	Period [ns]	14.9	18.2	18.5	18.2	20.5	23.5

Table 7: FPGA implementation results for polynomiers with 3-bit coefficients and estimations of the powers of x for $\sin(x)$ on $[0, \pi/4[$.

f	$\sin(x)$ on $[0, \pi/4[$		$\sqrt{(x)}$ on $[1, 2[$	
Degree	2	3	2	3
$w_I = w_O$	8	12	8	12
c	5	$3, 10$	5	$7, 7$
Area [# slices]	27	141	17	86
Period [ns]	16.7	28.5	17.2	29.4

Table 8: FPGA implementation results for other parameters or functions.

5 Comparison with Previous Work

We compare our results with two other methods dedicated to this range of precision: the multipartite tables from [1] and the single multiplication second order method (SMSO) from [2]. Those two references have been used because they provide complete results with

compatible FPGA targets. Our results are attractive compared to these methods but one should keep in mind that the accuracy targets are not the same. Methods from [1] and [2] provide faithfull rounding while our provides a very small average error but no faithfull rounding. So, our method is suitable for some applications in signal processing.

	Multipartite [1]		SMSO [2]		Our		
w_I	Area [# slices]	Period [ns]	Area [# slices]	Period [ns]	Area [# slices]	Period [ns]	d
8	19	16.6	21	8	27	14.9	2
12	76	18.0	63	14	50	20.5	3
16	280	24.8	123	19	71	23.5	3

Table 9: Comparison with methods from [1] and [2] for $\sin(x)$ on $[0, \pi/4[$.

An interesting method to compare with is the partial product arrays from [5]. Unfortunately, it seems that there is no FPGA implementation of this method. Shift-and-add algorithms, such as CORDIC [7], are not interesting for such low-precision implementations.

6 Conclusion and Future Prospects

We have presented a method for low-precision approximation of functions in hardware. The presented method leads to fast and small operators up to 16 bits of precision. The obtained operators provide very small average error with reasonable maximum error. This makes our method suitable for some applications in digital signal processing.

The proposed method is based on two modifications in the polynomial approximation. The first one is the use of quantified coefficients with up to 3 non-zero bits instead of full width coefficients. The second one is the use of estimations of the powers of x instead of the full width values of x^i. This leads to very small and fast circuits by replacing the costly multiplications by a small number of additions.

Compared to standard polynomiers, the proposed method shows very huge improvements. Our method provides up to 40% smaller solutions than the best literature results.

In a near future, we plan to work on ASIC targets as well as higher precision approximations (say up to 20 bits). We will focus on the study of the relations between the numerous parameters. We also plan to develop a tool which generates VHDL descriptions of polynomial evaluation circuits using our method.

Acknowledgements

This work was partially supported by an ACI grant from the French ministry of Research and Education.

References

[1] F. de Dinechin and A. Tisserand. Some improvements on multipartite tables methods. *IEEE Transactions on Computers*, 54(3):319–330, March 2005.

[2] J. Detrey and F. de Dinechin. Second order function approximation using a single multiplication on FPGAs. In *14th Intl Conference on Field-Programmable Logic and Applications*, pages 221–230, Antwerp, Belgium, August 2004. LNCS 3203.

[3] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann, 2003.

[4] M.D. Ercegovac, T. Lang, J.-M. Muller, and A. Tisserand. Reciprocation, square root, inverse square root, and some elementary functions using small multipliers. *IEEE Transactions on Computers*, 49(7):627–637, July 2000.

[5] H. Hassler and N. Takagi. Function evaluation by table look-up and addition. In S. Knowles and W.H. McAllister, editors, *12th IEEE Symposium on Computer Arithmetic*, pages 10–16. IEEE Computer Society Press, 1995.

[6] A. A. Liddicoat and M. J. Flynn. Parallel square and cube computations. In *34th Asilomar Conference on Signals, Systems, and Computers*, pages 1325–1329. IEEE, October 2000.

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

[8] J. A. Pineiro, J. D. Bruguera, and J.-M. Muller. Faithful powering computation using table look-up and a fused accumulation tree. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pages 40–47. IEEE Computer Society, 2001.

[9] E. Remes. Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. *C.R. Acad. Sci. Paris*, 198:2063–2065, 1934.

[10] M. Schulte and J. Stine. Approximating elementary functions with symmetric bipartite tables. *IEEE Transactions on Computers*, 48(8):842–847, August 1999.

[11] R. Stefanelli. A suggestion for a high-speed parallel binary divider. *IEEE Transactions on Computers*, 42(1):42–45, January 1972.

[12] D. A. Sunderland, R. A. Strauch, S. S. Wharfield, H. T. Peterson, and C. R. Role. CMOS/SOS frequency synthesizer LSI circuit for spread spectrum communications. *IEEE Journal of Solid State Circuit*, 19(4):497–506, August 1984.

[13] N. Takagi. Powering by a table look-up and a multiplication with operand modification. *IEEE Transactions on Computers*, 47(11):1216–1222, 1998.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (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