HAL
archives-ouvertes.fr

Automatic Generation of Low-Power Circuits for the Evaluation of Polynomials

Arnaud Tisserand

▶ **To cite this version:**

Arnaud Tisserand. Automatic Generation of Low-Power Circuits for the Evaluation of Polynomials. 40th Asilomar Conference on Signals, Systems and Computers, Oct 2006, Pacific Grove, CA (USA), IEEE, pp.2053-2057, 2006, <10.1109/ACSSC.2006.355128>. <lirmm-00125519>

HAL Id: lirmm-00125519

https://hal-lirmm.ccsd.cnrs.fr/lirmm-00125519

Submitted on 19 Jun 2007

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.

Automatic Generation of Low-Power Circuits for the Evaluation of Polynomials

Arnaud Tisserand
LIRMM, CNRS–Univ. Montpellier 2
161 rue Ada. F-34392 Montpellier, FRANCE
arnaud.tisserand@lirmm.fr

***Abstract*— This paper presents a method for the automatic generation of high-performance and low-power arithmetic operators based on polynomial approximations. It deals with the bit-level representation of the polynomial coefficients, the intermediate computations width, the approximation and the rounding errors. The generated operators are small, fast and numerically validated at design time. Some examples have been implemented on FPGAs.**

INTRODUCTION

The design of high-performance and pow-power arithmetic operators is an important issue in application specific integrated circuits (ASICs), systems on chip (SoCs) and field-programmable gate arrays (FPGAs) implementations. Basic operations such addition/subtraction or multiplication have always been implemented as high-performance operators in digital circuits [1]. Some recent applications require fast evaluation of more complex operations such as division, reciprocal, square-root, trigonometric functions, logarithm or exponential. Function approximation is often performed using polynomial evaluation in software as well as in hardware. For instance, elementary functions (sine, cosine, exponential, logarithm...) are often evaluated using polynomials [2].

This paper presents a method for the automatic generation of arithmetic operators based on polynomial approximations. This method has been presented in French in [3]. Here we deal its power consumption advantages. The proposed method produces polynomial approximations "well-suited" for high-performance hardware implementations. It faces with two problems: the generation of the polynomial coefficients that ensure low approximation errors and the sizing of intermediate computations that provide low round-off errors.

Notations and polynomial approximation background are presented in Section I. Some previous works are summarized in Section II. The proposed method is described in Section III and illustrated on some examples in Section IV.

I. NOTATIONS AND BACKGROUND

The target function is f with input and output in 2's complement fixed-point format. f is approximated using the degree-d polynomial p (no range reduction is considered here [2]). The argument x is in the domain $[a, b]$ and the result $p(x)$ is in the range $[a', b']$. Extension to other forms of input/output intervals is straightforward. The argument x is a n_x-bit number and the result $p(x)$ is a m-bit number as summarized in Figure 1. The input argument x is considered as exact. The position of the binary point is fixed by the format of the smallest representation that include both a and b. The number of fractional bits will be computed by the method to fit the target accuracy requirement. The polynomial coefficients are denoted p_0, p_1, p_2, ..., p_d, then $p(x) = \sum_{i=0}^{d} p_i x^i$. The coefficients p_i are represented using 2's complement or borrow-save [1] notations.

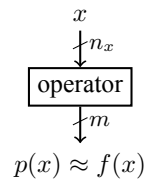

Fig. 1. Operator notations

Several evaluation schemes may be used to compute the value $p(x)$ in practice. In this work we only consider the *direct* and *Horner* evaluation schemes:

$$p(x) = \begin{cases} p_0 + p_1 x + p_2 x^2 + \cdots + p_d x^d & \text{direct} \\ p_0 + x\Big(p_1 + x\big(p_2 + x(\cdots + x p_d)\cdots\big)\Big) & \text{Horner} \end{cases}$$

Evaluation schemes differ on several aspects: computation cost, internal parallelism and accuracy. The direct scheme leads to a cost of d additions and $d + \lceil \log_2 d \rceil$ multiplications while the Horner scheme only requires d additions and d multiplications. The Horner scheme is a sequential structure while the direct scheme allows some internal parallelism. From the accuracy point of view, the Horner scheme is known to be slightly more accurate than the direct scheme.

The approximation to f using the polynomial p deals with two components: the *approximation error* and the *round-off error*. The approximation error measures the distance between the mathematical function f and the function used for the approximation, here p. The theoretical approximation error ϵ_{app} due to the use of the polynomial p to approximate the function f on $[a, b]$, is measured using:

$$\epsilon_{\text{app}} = ||f - p||_\infty = \max_{a \le x \le b} |f(x) - p(x)|. \qquad (1)$$

In Eq. (1), $p(x)$ is the mathematical value computed using an infinite precision. The approximation error ϵ_{app} is the smallest

theoretical error that can be obtained using the polynomial p for approximating f. Due to the finite precision of the coefficients and intermediate computations, we will have to deal with larger errors. ϵ_{app} can be numerically estimated using the Maple `infnorm` command.

Our polynomial approximations use the *minimax* polynomial as a starting point. The degree-d minimax polynomial approximation to f on $[a, b]$ is p^* which satisfies:

$$||f - p^*||_\infty = \min_{p \in \mathcal{P}_d} ||f - p||_\infty, \tag{2}$$

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 Remes [4], see also [2]. Here we use the Maple `minimax` command.

The *round-off error* or *rounding error* due to the finite precision of the intermediate and final values adds up to the approximation error. This error is small for one single operation, i.e. a fraction of the weight of the least significant bit (LSB). But during a sequence of operations, these small errors may accumulate themselves and significantly degrade the accuracy of the final result. In the following, errors are expressed directly or as equivalent accuracy. The accuracy is the number of correct or significant bits. The relation between the error ϵ and the accuracy μ is $\mu = -\log_2 |\epsilon|$.

The notation $()_2$ denotes the binary representation. We also use the *borrow-save* representation [1] denoted $()_{\text{bs}}$ (i.e. radix-2 redundant representation with digits in $\{-1 = \overline{1}, 0, 1\}$).

II. Summary of Previous Works

Here we summarize some previous works on arithmetic operators dedicated to function approximation in hardware. Detailed presentations may be found in [1] for computer arithmetic and in [2] for elementary functions.

A. Tables Based Methods

One of the first methods proposed to approximate functions was to tabulate the values of the target function for each possible input. This leads to exponential size tables: 2^{n_x} words of m bits. In practice the maximum number of address bits of the table n_x is in the range 8–12 depending on the technology.

Higher accuracies can be reached by combining tables and arithmetic operations. Architectures based on tables and additions/subtractions are very common for function approximation [5]. The bipartite method uses two tables and only one final addition as illustrated on Figure 2.

The bipartite method uses a decomposition of x into 3 subwords x_1, x_2 and x_3 of length n_1, n_2 and n_3 respectively such that $n_x = n_1 + n_2 + n_3$. The bipartite table method leads to tables with a total of only $2^{2n_x/3}$ address bits compared to the 2^{n_x} address bits of the single table solution. The bipartite table method has been extended to a number of tables larger than 2, it is called the multipartite method. It uses several tables looked-up in parallel and final addition of the all tables' contributions [5]. They allow computing commonly used functions with low accuracy (up to 24 bits) with significantly lower hardware cost than that of a straightforward table implementation, while being faster than digit-recurrence algorithms or basic polynomial approximations.

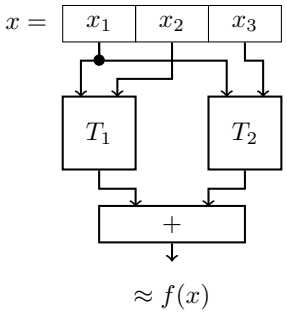

Fig. 2. Bipartite method architecture

B. Polynomial or Rational Approximations

The main drawback of table based methods is the fact that the tables are dedicated to only one specific function. Sharing tables among several functions is not possible in practice. Using polynomial or rational approximations, the basic operators (mainly adders and multipliers) required for the evaluation can be shared or reused. Function approximation in software is often performed using polynomial or rational approximations [2]. Rational approximations are not frequently used in hardware due to the high latency of the final division. Here we only deal with polynomial approximations.

Polynomial approximations are used in hardware since a long time [6]. The size of the multipliers is a major concern. Several solutions have been investigated to limit their size such as weighted sum methods [7]. The coefficients of the polynomial are distributed at the bit level. Thus the polynomial is rewritten as the sum of a huge set of weighted products of the bits of x. Some of these terms are neglected. Polynomial approximations are also combined with tables [8].

We will see below that the determination of polynomials with coefficients exactly representable in the target format is one of the main problems in the implementation of polynomial approximations. A general but quite slow method dedicated to this problem is proposed in [9].

C. Digit-Recurrence Algorithms

Digit-recurrence algorithms, also called *shift-and-add* algorithms, produce one digit of the result every iteration starting from the most significant digit (MSD) (e.g. the paper and pencil division method) [1]. The two most well known digit-recurrence algorithms are: the SRT algorithm for division, square root and other algebraic functions, and the CORDIC algorithm for elementary functions.

The SRT algorithm, [1], leads to a small number of iterations using a high radix. It uses a redundant number system for the result digits. This allows correcting some small errors from previous iterations due to the reduced internal precision. The combination of a redundant number system and reduced precision leads to very fast iterations.

The CORDIC algorithm (for COordinate Rotations on a DIgital Computer) only uses additions and shift to approximate some elementary functions, so it is very well suited for low-area hardware implementations. A complete description of this algorithm and its numerous variations can be found in [2].

III. Proposed Method

The proposed method has been presented in French in [3]. It provides an answer to two practical questions about implementation of polynomial approximation. The first one is "what values should be used for the implemented coefficients?". The second one is "what is the minimal size for the intermediate computations?".

The input parameters of the method are:

- f the function to be evaluated,
- $[a, b]$ the domain of the argument x,
- the argument format x (size n_x bits),
- μ the total maximal target absolute error (the accuracy constraint).

The results from the method are:

- d the degree of the polynomial,
- p_0, p_1, p_2,..., p_d the coefficients values (representable in the target format),
- n the coefficient size,
- n' the data-path size[1].

The proposed method consists in 3 main steps and an optional step detailed below.

A. Step 1: Determination of the Initial Polynomial

The first step defines a good starting point for the method. We use a minimax polynomial as a starting point (see Section I) provided by the Maple `minimax` function. We look for the minimax polynomial p^* with the smallest degree d and accurate enough to approximate f on $[a, b]$ with an error less than μ, i.e. $\epsilon^*_{\text{app}} < \mu$ with $\epsilon^*_{\text{app}} = ||f - p^*||_\infty$. We start with $d = 1$, and d is incremented until p^* leads to an approximation error ϵ^*_{app} such that $\epsilon^*_{\text{app}} < \mu$.

The first step provides: d the minimal degree of the approximation polynomial, $p^*(x) = \sum_{i=0}^{d} p_i^* x^i$ the theoretical polynomial (real coefficients), and ϵ^*_{app} the *minimal* theoretical error between the output of the circuit and f (using infinite accuracy for the coefficients and the computations).

The polynomial result p^* will be modified in the next steps. The next steps will degrade the accuracy (i.e. lead to errors larger than ϵ^*_{app}). Then some "margin" between ϵ^*_{app} and μ is necessary as seen in the next sections.

B. Step 2: Coefficients Optimization

In this step we look for the size n and the values p_i of the coefficients in the target format such that $\epsilon_{\text{app}} < \mu$ where $\epsilon_{\text{app}} = ||f - p||_\infty$ and $p(x) = \sum_{i=0}^{d} p_i x^i$. Notice that n may be smaller than the width of the target format.

[1]In some cases, using different values for n and n' may reduce the size of the circuit.

The proposed solution is based on an exploration over the rounded coefficients. Each coefficient p_i^* may be rounded up $p_i = \triangle(p_i^*)$ or down $p_i = \triangledown(p_i^*)$. There are 2 choices per coefficients, then 2^{d+1} different polynomials to test as illustrated in Figure 3. For each possible polynomial p the value ϵ_{app}. In our applications d is small ($d \leq 6$) then the total exploration time is small.

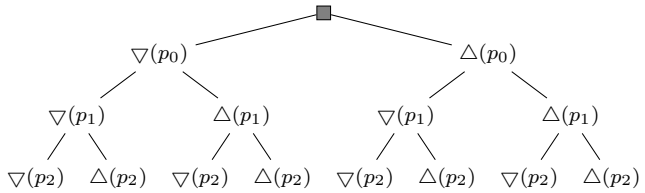

Fig. 3. Tested rounding modes for p^* of degree $d = 2$

We designed a Maple program that tests the 2^{d+1} polynomials corresponding to the different rounding modes of the $d+1$ coefficients p_i^*. The program starts with $n = \lceil -\log_2 |\mu| \rceil$, tests all the rounding modes of the $d + 1$ coefficients for the current n size. If there are solutions, i.e. polynomials such that $\epsilon_{\text{app}} < \mu$, it returns the list of those polynomials for the next step (those where ϵ_{app} is minimal). If there is no solution then n is incremented.

C. Step 3: Data-path Optimization

In this step we look for the data-path size n'. Only two polynomial evaluation schemes are supported in the method: the direct and the Horner schemes. The idea is very simple. We start with $n' = n$ and we check that the data-path of size n' fulfills the accuracy constraint μ using GAPPA. If the total error bound computed by GAPPA is less than μ then this step is finished. If not, the size n' is incremented and the new data-path should be tested using GAPPA.

The GAPPA software, presented in [10], allows to evaluate and to produce a proof of mathematical properties on numerical codes. The main useful characteristic of GAPPA used in this work is its capability to tightly bound round-off errors and prove these bounds are below some threshold. GAPPA can generate a file for the Coq proof assistant [11].

The difference between n' and n is called the number of *guard bits*. Using a coefficient size n smaller than n' allows a reduction in the memory size required to store the coefficients without degradation on the final accuracy.

D. Step 4 (optional): Post-Optimizations

One kind of frequent post-optimization is to simplify the hardware when some coefficients are very close to power of 2. As an example, if one coefficient from step 2 is the value $0.5002441406 = (0.100000000001)_2$ and the target accuracy μ is about 12 fractional bits, this coefficient may be rounded to 0.5. In that case, the multiplication by 0.5002441406 is replaced by a simple right shift, see Section IV-B for a concrete example of this kind of post-optimization.

E. Loops

From a given step, it may be necessary to move back to a previous step. For instance, there may be no result from a given step or its result may not be considered "good enough".

As an example, the second step may not produce representable coefficients that ensure $\epsilon_{\text{app}} < \mu$. This case is due to insufficient margin between ϵ^*_{app} and μ in the first step. In that case, one should move back to step 1 and try with a higher degree polynomial (i.e. $d \leftarrow d+1$).

Another loop occurs after step 3 when n' is considered too huge in front of n. Then it may be more efficient to move back to step 2 and try a larger n. By doing this, ϵ_{app} will be smaller which leads to more margin for round-off errors.

IV. Examples

The examples below have been implemented on Xilinx FPGAs XCV200-5 using ISE8.1i tools. Synthesis and place/route have been optimized for an area target with high effort. The reported results include all the resources required for the implementation (logic cells and registers). The power consumption is estimated using the XPower software and solutions are compared on the same set of random input patterns.

A. Radix-2 Exponential over $[0,1]$

Here $f(x) = 2^x$ and $x \in [0,1]$. The target accuracy is 12 bits, i.e. $\mu = 2^{-12}$. We report the result from the step 1 for the minimax polynomial for $1 \leq d \leq 5$. The corresponding accuracy is reported below in number of correct bits.

d	1	2	3	4	5
ϵ^*_{app}	4.53	8.65	13.18	18.04	23.15

Degree-1 and -2 theoretical minimax polynomials are not accurate enough with respect to the 12-bit target accuracy. Without the presented generation method a degree-4 polynomial would be required assuming worst case round-off error. Indeed for degree 3 one have $13.18-d = 10.18 < 12$ while for degree 4 one have $18.04 - d = 14.04 > 12$. The values below represent all the rounding modes for the degree-4 solution and their accuracy. One can notice the large variation of the approximation errors for the rounding modes of the theoretical coefficients from 11.41 to 17.12 bits of accuracy.

$(\nabla,\nabla,\nabla,\nabla,\nabla)$	12.00	$(\nabla,\nabla,\nabla,\nabla,\triangle)$	13.00
$(\nabla,\nabla,\nabla,\triangle,\nabla)$	13.00	$(\nabla,\nabla,\nabla,\triangle,\triangle)$	14.03
$(\nabla,\nabla,\triangle,\nabla,\nabla)$	13.00	$(\nabla,\nabla,\triangle,\nabla,\triangle)$	14.55
$(\nabla,\nabla,\triangle,\triangle,\nabla)$	14.99	$(\nabla,\nabla,\triangle,\triangle,\triangle)$	13.00
$(\nabla,\triangle,\nabla,\nabla,\nabla)$	13.00	$(\nabla,\triangle,\nabla,\nabla,\triangle)$	**16.13**
$(\nabla,\triangle,\nabla,\triangle,\nabla)$	**17.12**	$(\nabla,\triangle,\nabla,\triangle,\triangle)$	13.00
$(\nabla,\triangle,\triangle,\nabla,\nabla)$	15.71	$(\nabla,\triangle,\triangle,\nabla,\triangle)$	13.00
$(\nabla,\triangle,\triangle,\triangle,\nabla)$	13.00	$(\nabla,\triangle,\triangle,\triangle,\triangle)$	12.00
$(\triangle,\nabla,\nabla,\nabla,\nabla)$	13.00	$(\triangle,\nabla,\nabla,\nabla,\triangle)$	13.00
$(\triangle,\nabla,\nabla,\triangle,\nabla)$	13.00	$(\triangle,\nabla,\nabla,\triangle,\triangle)$	13.00
$(\triangle,\nabla,\triangle,\nabla,\nabla)$	13.00	$(\triangle,\nabla,\triangle,\nabla,\triangle)$	13.00
$(\triangle,\nabla,\triangle,\triangle,\nabla)$	12.99	$(\triangle,\nabla,\triangle,\triangle,\triangle)$	12.00
$(\triangle,\triangle,\nabla,\nabla,\nabla)$	12.99	$(\triangle,\triangle,\nabla,\nabla,\triangle)$	12.98
$(\triangle,\triangle,\nabla,\triangle,\nabla)$	12.91	$(\triangle,\triangle,\nabla,\triangle,\triangle)$	12.00
$(\triangle,\triangle,\triangle,\nabla,\nabla)$	12.79	$(\triangle,\triangle,\triangle,\nabla,\triangle)$	12.00
$(\triangle,\triangle,\triangle,\triangle,\nabla)$	12.00	$(\triangle,\triangle,\triangle,\triangle,\triangle)$	11.41

Using our method we test the degree-3 solution. In that case the theoretical minimax polynomial is $p^*(x) = 0.9998929656 + 0.6964573949x + 0.2243383647x^2 + 0.0792042402x^3$. The approximation error is $\epsilon_{\text{app}} = ||f - p^*||_\infty = 0.0001070344$, this corresponds to 13.18 bits of accuracy (assuming infinite precision for the p_i and all computations). We consider a fixed-point format with one integer bit and $n-1$ fractional bits. We report below the results from step 2 for several values of $n-1$:

$n-1$	12	13	14	15	16
ϵ_{app}	12.38	12.45	13.00	13.00	13.02
# step 2 results	0	0	2	2	7

For the solution $n-1 = 14$ bits, all the rounding modes possible in step 2 are reported below:

$(\nabla,\nabla,\nabla,\nabla)$	11.41	$(\nabla,\nabla,\nabla,\triangle)$	12.00
$(\nabla,\nabla,\triangle,\nabla)$	12.00	$(\nabla,\nabla,\triangle,\triangle)$	12.84
$(\nabla,\triangle,\nabla,\nabla)$	12.00	$(\nabla,\triangle,\nabla,\triangle)$	**13.00**
$(\nabla,\triangle,\triangle,\nabla)$	**13.00**	$(\nabla,\triangle,\triangle,\triangle)$	12.36
$(\triangle,\nabla,\nabla,\nabla)$	12.00	$(\triangle,\nabla,\nabla,\triangle)$	12.25
$(\triangle,\nabla,\triangle,\nabla)$	12.23	$(\triangle,\nabla,\triangle,\triangle)$	12.23
$(\triangle,\triangle,\nabla,\nabla)$	12.13	$(\triangle,\triangle,\nabla,\triangle)$	12.12
$(\triangle,\triangle,\triangle,\nabla)$	12.05	$(\triangle,\triangle,\triangle,\triangle)$	11.64

There are two solutions: $\frac{8191}{8192} + \frac{2853}{4096}x + \frac{1837}{8192}x^2 + \frac{649}{8192}x^3$ and $\frac{8191}{8192} + \frac{2853}{4096}x + \frac{919}{4096}x^2 + \frac{649}{8192}x^3$. Those polynomials lead to an approximation error of 0.0001220703 (13.00 correct bits).

In step 3, we look for the data-path size of the operator. We report below the final accuracy evaluated using GAPPA for the first possible polynomial and for several sizes n':

n'	14	15	16	17	18	19
Horner	11.32	11.93	12.36	12.65	12.81	12.90
direct	11.24	11.86	12.32	12.62	12.79	12.89

The values obtained for the other polynomial ($p_2 = \frac{919}{4096}$) are similar. Two solutions have been implemented. The first one corresponds to the standard solution (degree-4 polynomial, $n = n' = 18$). The second one is the result from the generation method ($d = 3$, $n-1 = 14$, Horner evaluation scheme and $n' = 16$). The implementation results are reported in Table I. The generation method leads to 17% smaller circuit and the degree-3 approximation saves 38% of the computation time.

solution	area [slices]	period [ns]	#cycles	delay [ns]	relative power
$d=3$, $n'=16$	193	21.9	3	65.7	1.00
$d=4$, $n'=18$	233	26.9	4	107.6	0.68

TABLE I
IMPLEMENTATION RESULTS 2^x OVER $[0,1]$

B. Square Root

Here $f(x) = \sqrt{x}$, $x \in [1,2]$ and a target accuracy of 8 correct bits is fixed ($\mu = 2^{-8}$). The first step leads to $d = 2$ and $\epsilon^*_{\text{app}} = 0.0007638369$ which corresponds to 10.35 correct bits (for $d = 1$ the accuracy is only 6.81 correct bits).

The minimax polynomial is $p^* = 0.4456804579 + 0.6262821240x - 0.7119874509x^2$. The fixed-point evaluation of this polynomial requires some scaling in the fixed-point format. With x in $[1, 2]$, 2 integer bits are required for x^2 while the other operations only require 1 integer bit. In order to avoid this scaling problem, we now consider $f(x) = \sqrt{1+x}$ with $x \in [0, 1]$ (this is the exactly the same function). Then the minimax polynomial is: $1.0007638368 + 0.4838846338x - 0.7119874509x^2$. The theoretical approximation error also leads to 10.35 correct bits (the variable change $x = 1 + x$ does not modify the minimax polynomial quality).

The coefficients p_0 and p_1 are close to power of 2 and we will try to use the post-optimizations proposed in step 4 (see Section III-D). The first optimization consists in using $p_0 = 1$. The approximation polynomial $1 + 0.4838846338x - 0.7119874509x^2$ leads to a theoretical accuracy of 9.35 correct bits. The coefficient p_1 is close to 0.5. The approximation $1 + 0.5x - 0.7119874509x^2$ only leads to 6.09 correct bits. So p_1 can not be replaced by 0.5. But we can try to recode p_1 with only a few non-zero digits (i.e. 1 or $-1 = \overline{1}$). The coefficient p_1 is close to $(0.100001)_2$. The approximation polynomial $1 + (0.100001)_2x - 0.7119874509x^2$ leads to an accuracy of 9.45 correct bits and the product p_1x is replaced by the subtraction $\frac{1}{2}x - \frac{1}{2^6}x$. We try to recode p_2 using a few non-zero bits and we get $p_2 = (0.0001001)_2$. The product p_2x^2 is replaced by the addition $\frac{1}{2^4}x^2 + \frac{1}{2^7}x^2$.

The approximation polynomial $1 + (0.100001)_2x + (0.0001001)_2x^2$ leads to a accuracy of 9.49 bits (without round-off errors). GAPPA returns a total of 8.07 correct bits for $n' = 13$ with only one multiplication x^2 as illustrated in Figure 4 (gray circles denotes right shifts).

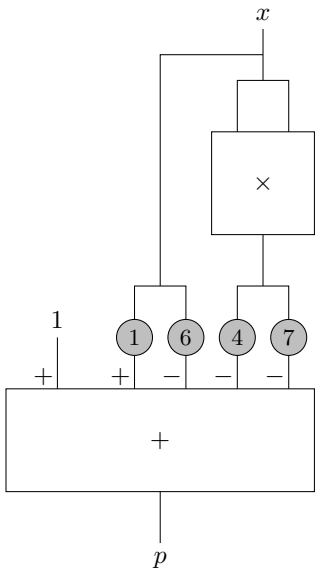

Fig. 4. Post-optimized operator for $\sqrt{1+x}$ over $[0, 1]$

The operator presented in Figure 4 and the standard solution (degree 2, Horner scheme with $n = n' = 11$ bits) have been implemented on FPGAs. The results are reported in Table II. A 40% area reduction and a 51% speedup are obtained.

solution	area [slices]	period [ns]	#cycles	delay [ns]	relative power
$d = 2$ Horner	103	19.9	2	39.8	1.00
$d = 2$ optimized	61	19.4	1	19.4	0.45

TABLE II
IMPLEMENTATION RESULTS $\sqrt{1+x}$ OVER $[0, 1]$

CONCLUSION

A method based on polynomial approximation was described. This method leads to small, fast and low-power hardware operators. The generated operators are numerically validated to design time. The method deals with both the approximation and the round-off errors which is a something new. This method does not explore all the parameter space corresponding to these two questions, the result may not be optimal. At the theoretical point of view, the optimal result is not known! Some have been implemented, validated and compared to standard solutions. Some significant improvements are reported: up to 40% for circuit area reduction, up to 50% speed improvement and up to 55% power reduction.

REFERENCES

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

[2] J.-M. Muller. *Elementary Functions: Algorithms and Implementation*. Birkhäuser, 2nd edition, 2006.

[3] R. Michard, A. Tisserand, and N. Veyrat-Charvillon. Optimisation d'opérateurs arithmétiques matériels à base d'approximations polynomiales. In *11ième SYMPosium en Architectures nouvelles de machines (SYMPA)*, pages 130–141, Perpignan, France, October 2006.

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

[5] F. de Dinechin and A. Tisserand. Multipartite table methods. *IEEE Transactions on Computers*, 54(3):319–330, March 2005.

[6] W. P. Burleson. Polynomial evaluation in VLSI using distributed arithmetic. *IEEE Transactions on Circuits and Systems*, 37(10), 1990.

[7] K. Johansson, O. Gustafsson, and L. Wanhammar. Approximation of elementary functions using a weighted sum of bit-products. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 795–798. IEEE, May 2006.

[8] J. Detrey and F. de Dinechin. Table-based polynomials for fast hardware function evaluation. In *Proc. 16th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 328–333, Samos, Greece, July 2005. IEEE Computer Society.

[9] N. Brisebarre, J.-M. Muller, and A. Tisserand. Computing machine-efficient polynomial approximations. *ACM Transactions on Mathematical Software*, 32(2):236–256, June 2006.

[10] G. Melquiond. GAPPA: génération automatique de preuves de propriétés arithmétiques. http://lipforge.ens-lyon.fr/www/gappa/, 2006. LIP, ENS-Lyon.

[11] The Coq Development Team. The Coq proof assistant. http://coq.inria.fr/, 2004. INRIA.