

# Function Evaluation on FPGAs using On-Line Arithmetic Polynomial Approximation

Rachid Beguenane\*, Stéphane Simard\* and Arnaud Tisserand†

\* ERMETIS, Univ. Québec at Chicoutimi

555, boulevard de l'Université. Chicoutimi (QC), G7H 2B1, CANADA

rbeguena@uqac.ca s.simard@ieee.org

† LIRMM, CNRS–Univ. Montpellier II

161 rue Ada. F-34392 Montpellier, FRANCE

arnaud.tisserand@lirmm.fr

**Abstract**—This paper presents the first results of a young collaboration between ERMETIS and LIRMM on hardware arithmetic operators for digital control and digital signal processing. It presents on-line arithmetic operators for the polynomial approximation of some functions (e.g., reciprocal, square-root, sine, cosine, exponential, logarithm). The proposed method is based on polynomial approximations with sparse coefficients well suited for FPGA implementation.

## I. INTRODUCTION

Digit-serial arithmetic is often proposed to implement algorithms in digital signal processing and digital control applications [1]. Its main advantages are: small size operators, small number of communication lines and overlapping of consecutive operations (pipeline). Digit-level pipeline and operation parallelism (enabled by the small size of the operators) may compensate for the low speed inherent to the digit serial transmission.

In digit-serial operators, the transmission may begin with the *least significant digit first* (LSDF) or with the *most significant digit first* (MSDF). Recent works use LSDF arithmetic in a more a less systolic architecture [2]. In practice the MSDF mode, or *on-line arithmetic*, has several advantages:

- accurate analog to digital converters (ADCs) only work in the MSDF mode;
- all operations can be computed MSDF using a redundant number system (comparison and division cannot be computed directly in the LSDF mode);
- simpler accuracy management (the significant and useful digits are produced in the beginning).

Many algorithms and implementations have been proposed for the main operations such as addition, multiplication, division and square root in on-line arithmetic [3, chap. 9]. But for function evaluation (e.g., sine, cosine, exponential, logarithm), there are only a few practical results [4]. The purpose of this paper is the evaluation of functions using polynomial approximations with sparse coefficients on FPGAs. The specific formulation of the polynomial evaluation leads to a very simple architecture well suited for FPGAs.

This paper is organized as follows. Section II presents a short introduction to on-line arithmetic. Section III recalls basic notions on polynomial approximation. The proposed

method is presented in Section IV for the evaluation of one function. The method is illustrated in the case of multiple functions in Section V. Conclusion is presented in Section VI.

## II. BACKGROUND ON ON-LINE ARITHMETIC

On-line arithmetic was introduced in 1977 by Ercegovac and Trivedi [5]. In on-line arithmetic the operands as well as the results flow in a digit-serial fashion with MSDF through arithmetic units. Algorithms for the main operations can be found in [3, chap. 9] and [6]. In the following, we will use the radix-2 addition, multiplication and square algorithms from these references. Important timing characteristics of on-line operators are their *on-line delay* ( $\delta$ ) and *period* ( $\tau$ ), as illustrated in Fig. 1. The on-line delay  $\delta$  corresponds to the latency of the operator (i.e.,  $\delta$  digits of the input(s) are required to compute the first digit of the result).

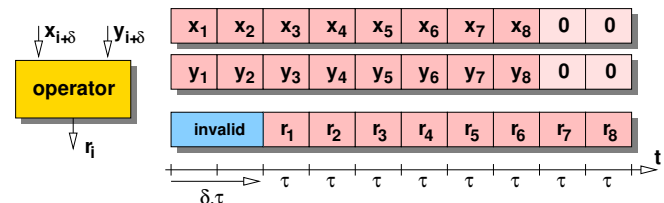


Fig. 1. On-Line Operator Timing Characteristics

In order to allow all computations in the MSDF mode, on-line arithmetic uses a redundant number system [3, chap. 9]. In this work, we focus on radix-2 algorithms using the *borrow-save* (signed) representation and a fixed-point format. In the borrow-save representation, the value  $x$  is represented by  $\sum_{i=-n}^{-m} x_i 2^{-i}$  with  $x_i \in \{-1, 0, 1\}$ . The number of integer bits  $m$  is very small in our target applications, the accuracy is represented by the number of fractional bits  $n$ . The developed method provides approximations up to 16 bits of accuracy. The notation  $()_2$  denotes the binary representation of a value. Bits with a negative weight are denoted by  $\bar{1}$ .

In the first phase of our collaboration, we use the results from the BigSky environment [7] and the article [6]. In a near future we plan to develop our own library of on-line arithmetic operations in order to measure the impact of the radix and digit

coding. BigSky generates a description of networks of on-line arithmetic circuits based on a library of basic operations. Table I presents estimations of the main characteristics of the on-line operators used below. The parameter  $N$  is the total number of digits (i.e.,  $N = n + m$ ). These values are extracted (or extrapolated for those marked with  $\approx$ ) from [7] and [6]. The implementations results from [7] have been obtained on Virtex FPGAs from Xilinx (small device, lowest speed grade, no package impact).

operation	# inputs	area # slices	speed MHz	on-line delay
addition	2	3	253	2
addition	$k$	$2k - 2$	n.a.	$\lceil \log_2 k \rceil + 1$
multiplication	2	$7N + 1$	101	3
mult. by cst.	1	$\approx 4N + 3$	$\approx 140$	2
square	1	$5N + 3$	135	3

TABLE I

ESTIMATION OF THE MAIN CHARACTERISTICS OF SOME ON-LINE OPERATORS (EXTRACTED OR EXTRAPOLATED FROM [7] AND [6])

### III. BACKGROUND ON POLYNOMIAL APPROXIMATION

In this work we use polynomial approximations to evaluate functions. A detailed presentation of polynomial approximations and other function evaluation methods can be found in [8]. Function  $f$  is evaluated with the argument  $x$  in the domain  $[a, b]$ . The input  $x$  and the result  $f(x)$  are in fixed-point format. The integer  $d$  denotes the degree of the approximation polynomial  $P$ . In our current applications, the approximation of standard functions require polynomials up to degree 5 (less than 24 bits of accuracy on small intervals).

In order to measure the theoretical approximation error  $\epsilon_{th}$  due to the use of the polynomial  $P$  to evaluate the function  $f$  on  $[a, b]$ , we use the distance (estimated using the Maple `infnorm` function):

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

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 Remes [9] (implemented in the Maple `minimax` function).

### IV. PROPOSED METHOD

When using polynomial approximations, the size of the multipliers is the main limitation at the circuit level. We propose to replace some large multiplications by additions thanks to polynomial approximations with sparse coefficients (i.e., with a lot of 0s). In order to illustrate our method,

we will use an example in the following:  $f(x) = \cos(x)$ ,  $x \in [0, \pi/4]$ ,  $N = n = 16$  bits,  $m = 0$  and  $d = 3$ , so  $P(x) = p_0 + p_1x + p_2x^2 + p_3x^3$  and  $x = 0.x_1x_2 \dots x_{16}$ .

The starting point of our method is the best polynomial approximation to  $f$  using the minimax polynomial  $P_{th}$ . For the cosine example, Maple provides the minimax polynomial:

$$P_{th} = 0.999886 + 0.004690x - 0.530309x^2 + 0.063046x^3.$$

This polynomial leads to a theoretical approximation accuracy of 13.10 bits ( $\epsilon_{th} = 0.0001135$ ).

Since the constant coefficient  $p_0$  is not involved in any multiplication, there is no need to convert it to a sparse coefficient. The integer  $\alpha_i$  denotes the number of non-zero bits in coefficient  $p_i$ .

The conversion of the minimax coefficients to sparse coefficients is performed using a simple iterative algorithm. At each iteration, the power of 2 the nearest to the coefficient is subtracted or added (depending on its sign) to the coefficient. This iteration is applied to the remainder until the target precision is reached or enough digits have been used (maximal value allowed for  $\alpha_i$ ).

For a given  $n$ -digit format (relative error  $2^{-n}$ ) and at most  $\alpha_i$  non-zero digits, this algorithm provides the *canonical signed digit* (CSD) representation of the value truncated to  $n$  digits with  $\alpha_i$  digits. This representation ensures that the number of non-zero digits in the recoded value is minimal. Using this recoding on a  $n$ -bit unsigned value, the number of non-zero digits is bounded by  $(n + 1)/2$  and it tends asymptotically to an average value of  $n/3 + 1/9$ , as shown in [10].

Table II presents the conversion result of the coefficient  $p_2 = -0.530309$  to sparse values in the CSD representation for several values of  $\alpha_2$ . The last line corresponds to the standard binary value (b.v.) of  $|p_2|$ .

$\alpha_2$	result	accuracy (# bits)
1	$(0.\overline{1}00000000000000)_2$	5.04
2	$(0.\overline{1}000\overline{1}0000000000)_2$	10.05
3	$(0.\overline{1}000\overline{1}0000100000)_2$	14.78
4	$(0.\overline{1}000\overline{1}000010000\overline{1}0)_2$	17.60
b.v.	$-(0.1000011111000010)_2$	17.60

TABLE II

CSD CONVERSION OF  $p_2$  FOR SEVERAL VALUES OF  $\alpha_2$

We denote by  $(\alpha_1, \alpha_2, \dots, \alpha_d)$  the *decomposition* with coefficient  $p_i$  on  $\alpha_i$  non-zero bits for all values of  $i$  (all the monomials  $p_i x^i$ ,  $p_0$  is not converted). There are many solutions for the decomposition of the  $d$  coefficients of polynomial  $P$ . The question is what is the best one? In order to answer to this question we propose to explore all the possible decompositions. This exhaustive search is feasible in practice because of the small size of our exploration space.

For each coefficient  $p_i$ , there is no need to explore decompositions with an accuracy better than the format accuracy ( $n$ -bit

computations). This means that  $\alpha_i$  is bounded by the maximum number of non-zero digits in the CSD representation. By definition, the CSD representation ensures  $\alpha_i \leq (n+1)/2$  for  $n$ -bit values [10]. In our target applications, the operand size is up to 24 bits.

For each monomial  $p_i x^i$ ,  $\alpha_{max,i}$  denotes the maximum value of  $\alpha_i$  for  $p_i$  with respect to the format accuracy. Therefore, the total number of decompositions is  $\prod_{i=1}^d \alpha_{max,i}$ . In our target applications,  $d$  is limited to 5. So, in practice, the maximal number of decompositions to test is  $(24/2)^5 = 248832$ . All those tests are independent, so they can be performed in parallel in order to speed up the process. In the cosine example, we have  $\alpha_{max,1} = \alpha_{max,2} = \alpha_{max,3} = 4$ . Then the number of decompositions to test is only  $4^3 = 64$ .

The polynomial corresponding to decomposition  $(\alpha_1, \alpha_2, \dots, \alpha_d)$  is denoted  $P_{(\alpha_1, \alpha_2, \dots, \alpha_d)}$ . The quality of this polynomial approximation is measured using its accuracy ( $\|f - P_{(\alpha_1, \alpha_2, \dots, \alpha_d)}\|_\infty$  on  $[a, b]$ ) and its evaluation cost. In a first time, we will evaluate the cost using the number of terms in the decomposition:  $1 + \sum_{i=1}^d \alpha_i$ .

Figure 2 presents the accuracy and the cost of the 64 decompositions of the cosine example. The dashed line represents the accuracy of the minimax polynomial (13.10 bits). This kind of figure may help to determine what is the best tradeoff between accuracy and cost with respect to the application constraints. In Figure 2, six decompositions have been selected and are detailed in Table III.

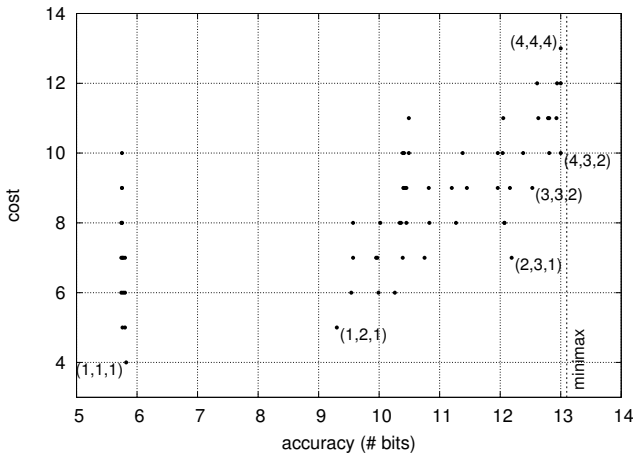


Fig. 2. Accuracy and Cost of the 64 Decompositions of the Cosine Example

The decomposition with the highest accuracy (first comparison criterion) and the smallest cost (second comparison criterion) is (4,3,2) (our . The coefficients of  $P_{(4,3,2)}$  are:  $p_0 = (0.111111111111000)_2$ ,  $p_1 = (0.0000000100110100)_2$ ,  $p_2 = (0.1000100001000000)_2$  and  $p_3 = (0.0001000000100000)_2$ .

The architecture for the evaluation of this polynomial is depicted in Figure 3. The power stage generates the powers of  $x$  required in the polynomial evaluation. In this example  $x^2$  and  $x^3$  are generated. The FIFOs in this stage are used to compensate for the on-line delay of the operators. The area

decomp.	accur.	cost	polynomial
(1,1,1)	5.82	4	$\frac{8191}{8192} + \frac{1}{256}x - \frac{1}{2}x^2 + \frac{1}{16}x^3$
(1,2,1)	9.30	5	$\frac{8191}{8192} + \frac{1}{256}x - \frac{17}{32}x^2 + \frac{1}{16}x^3$
(2,3,1)	12.19	7	$\frac{8191}{8192} + \frac{5}{1024}x - \frac{545}{1024}x^2 + \frac{1}{16}x^3$
(3,3,2)	12.53	9	$\frac{8191}{8192} + \frac{19}{4096}x - \frac{545}{1024}x^2 + \frac{129}{2048}x^3$
(4,3,2)	13.00	10	$\frac{8191}{8192} + \frac{77}{16384}x - \frac{545}{1024}x^2 + \frac{129}{2048}x^3$
(4,4,4)	13.00	13	$\frac{8191}{8192} + \frac{77}{16384}x - \frac{17377}{32768}x^2 + \frac{16527}{262144}x^3$

TABLE III

SELECTED DECOMPOSITIONS FROM THE COSINE EXAMPLE

cost and on-line delay of the power stage may be estimated using the values from Table I. In a future work, we will provide an optimized power stage with a its cost and on-line delay estimations.

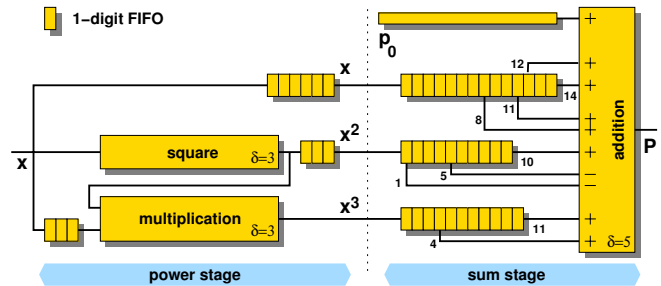


Fig. 3. Architecture of Decomposition (4,3,2) in the Cosine Example

The sum stage uses the decomposition characteristics. The monomial  $p_3 x^3$  is decomposed in  $(0.0001000000100000)_2 x^3 = 2^{-4}x^3 + 2^{-11}x^3$ . The other decompositions are:  $p_2 x^2 = -2^{-1}x^2 - 2^{-5}x^2 + 2^{-10}x^2$  and  $p_1 x = 2^{-8}x + 2^{-11}x + 2^{-12}x + 2^{-14}x$ . The constant coefficient  $p_0$  is stored in a specific shift register. All the terms of the decomposition are added (or subtracted) to produce  $P(x)$ .

The numerous FIFOs used in the architecture presented in Figure 3 do not represent a huge area in the FPGA. We use the shift registers of the slice in Virtex FPGAs. In practice any FIFO up to 16 borrow-save digits can be implemented in only one slice! So all the FIFOs of Figure 3 only use 6 slices (can be optimized to 5 by merging the two FIFOs of the  $x^2$  line since  $10 + 3 < 16$ ). So the cost is dominated by the power stage. But for given values of  $d$  and  $n$ , this stage has a fixed cost. This is why we use  $1 + \sum_{i=1}^d \alpha_i$  as a cost estimation in a first approximation. The use of more complex solutions for the sharing of the powers of  $x$  delay lines are considered for further work.

Based on the estimation of the operators cost from Table I, we explore the cosine example with  $d \in \{2, 3, 4\}$  for the FPGA cost (estimations in number of Virtex slices). The results of this exploration are presented in Figure 4.

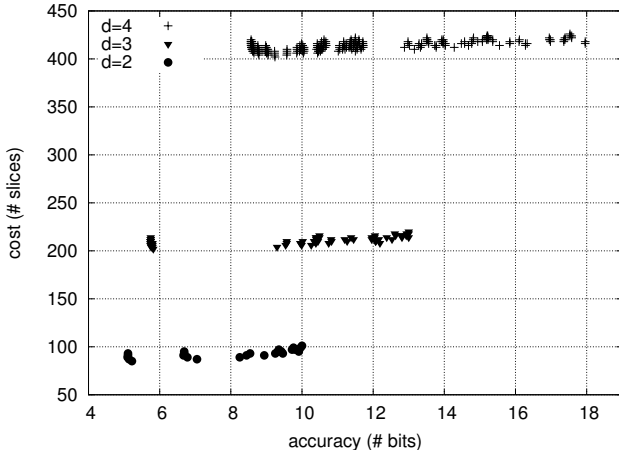


Fig. 4. Cosine Accuracy and Cost for  $d = 1$ ,  $d = 2$  and  $d = 3$

## V. PROPOSED METHOD FOR THE EVALUATION OF MULTIPLE FUNCTIONS

The architecture depicted in Figure 3 suggests that only the sum stage is specific to a function. The power stage can be shared for the evaluation of multiple functions. This principle is illustrated on Figure 5.

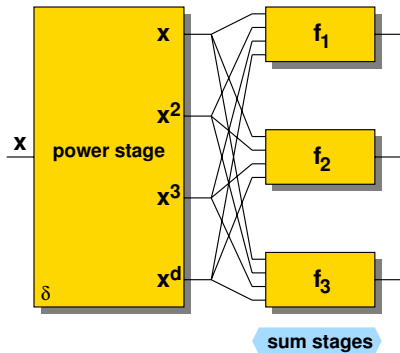


Fig. 5. Sharing of the Power Stage for Multiple Functions Operators

As an example, we use the functions  $\sin$  and  $\cos$  on  $x \in [0, \pi/4]$  with  $n = 16$  bits and  $d = 3$ . For the cosine function, we use the decomposition (4,3,2) from Section IV. The method for the sine function gives the best decomposition (2,2,3), it leads to 13.73 bits of accuracy and a cost of 210 slices. We also implemented a merged version for both functions.

Table IV clearly shows the benefit of the method for the evaluation of multiple functions. The cost of the “merged” version (cos, sin) is only 14% larger than the cost a single function operator. This property seems very interesting for multiple input multiple output (MIMO) systems.

$f$	cos	sin	(cos, sin)
cost	214	210	239

TABLE IV  
EVALUATION OF cos AND sin

Some small area improvement can be obtained by sharing common subexpressions in the multiple sum stages. Another improvement way is to use the tool presented in [11] in order to share coefficients or part of the coefficients in several polynomials.

## VI. CONCLUSION

This paper presents a method for the evaluation of functions based on on-line arithmetic polynomial approximations with sparse coefficients. This method seems to be very interesting for multiple functions evaluation.

In a near future, we plan to develop a tool to automatically generate the corresponding operators. We also plan to measure the impact of the choice of the radix and the digit coding on the cost and speed of the operators.

At a theoretical level, our method only deals with the approximation error. We plan to try to integrate in our method the evaluation error due to round off.

## REFERENCES

- [1] R. Hartley and P. Corbett, “Digit-serial processing techniques,” *IEEE Transactions on Circuits and Systems*, vol. 37, no. 6, pp. 707–719, June 1990.
- [2] A. Aggoun, M. Ibrahim, and A. Ashur, “Bit-level pipelined digit-serial array processors,” *IEEE Transactions on Circuits and Systems II*, vol. 45, no. 7, pp. 857–868, July 1998.
- [3] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2003.
- [4] J.-L. Beuchat and A. Tisserand, “Évaluation polynomiale en-ligne de fonctions élémentaires sur FPGA,” *Technique et Science Informatiques*, vol. 23, no. 10, pp. 1247–1267, 2004.
- [5] K. S. Trivedi and M. D. Ercegovac, “On-line algorithms for division and multiplication,” *IEEE Transactions on Computers*, vol. 26, no. 7, pp. 681–687, July 1977.
- [6] 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*, vol. 22, no. 2, pp. 336–345, Aug. 1994.
- [7] A. Schneider, R. McIlhenny, and M. D. Ercegovac, “BigSky—an on-line arithmetic design tool for FPGAs,” in *Proc. of the 8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa Valley, CA, USA, Apr. 2000, pp. 303–304.
- [8] J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, 2nd ed. Birkhäuser, 2006.
- [9] E. Remes, “Sur un procédé convergent d’approximations successives pour déterminer les polynômes d’approximation,” *C.R. Acad. Sci. Paris*, vol. 198, pp. 2063–2065, 1934.
- [10] R. I. Hartley, “Subexpression sharing in filters using canonic signed digit multipliers,” *IEEE Transactions on Circuits and Systems II*, vol. 43, no. 10, pp. 677–688, Oct. 1996.
- [11] N. Brisebarre, J.-M. Muller, and A. Tisserand, “Computing machine-efficient polynomial approximations,” *ACM Transactions on Mathematical Software*, 2006, to appear.