Evaluation of elementary functions without range reduction

Filipe A. Meireles[a], António J. Araújo[b]

[a]Faculdade de Engenharia da Universidade do Porto
[b]INESC Porto, Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, 4200–465 Porto, Portugal

ABSTRACT

The evaluation of elementary functions can be performed by approximations using minimax polynomials requiring simple hardware resources. The general method to calculate an elementary function is composed by three steps: range reduction, computation of the polynomial in the reduced argument and range reconstruction. This approach allows a low-degree polynomial approximation but range reduction and reconstruction introduce a computation overhead.

This work proposes an evaluation methodology without range reduction and range reconstruction steps. Applications that need to compute elementary functions may benefit from avoiding these steps if the argument belongs to a sub-domain of the function. Particularly in the context of embedded systems, applications related to digital signal processing most of the times require function evaluation within a specific interval. As a consequence of not doing range reduction, the degree of the approximant polynomials increases to maintain the required precision. Interval segmentation is an effective way to overcome this issue because the approximations are computed in smaller intervals. The proposed methodology uses non-uniform segmentation as a way to mitigate the problem arising from not carrying out range reduction. The benefits that come from applying interval segmentation to the general evaluation technique are limited by the range reduction and reconstruction steps because the segmentation only applies to the approximation step. However, when used in the proposed methodology it reveals more effective.

Some elementary functions were implemented using the proposed methodology in a FPGA device. The metric used to characterize the proposed technique are the area occupation and the corresponding latency. The results of each implementation without range reduction were compared with the corresponding ones of the general method using range reduction. The results show that latency can be significantly reduced while the area is approximately the same.

Keywords: Elementary functions, range reduction, polynomial approximation, FPGA

1. INTRODUCTION

Function evaluation can often be the performance bottleneck of many important compute-bound applications. Elementary mathematical functions, such as trigonometric, logarithmic, square root, reciprocal, and combinations of these functions, are extensively used in computer graphics, digital signal processing, robotics, astrophysics, fluid dynamics, etc. Computing these functions quickly and accurately is a major goal in computer arithmetic and hardware design in general. Software implementations are often too slow for numerically intensive or real-time applications. For instance, it is reported[1] that over 60% of the total runtime is spent on function evaluation operations in a simulation of a jet engine. The performance of such applications depends on the design of an efficient hardware function evaluator. Advanced FPGAs enable the development of low-cost and high-speed function evaluation units, customizable to particular applications. Yet, in order to implement function evaluation efficiently, the hardware designer is faced with a multitude of function evaluation methods.

To compute elementary functions, iterative algorithms, such as the CORDIC (COordinate Rotation DIgital Computer) algorithm,[2,3] have been often used. Although the CORDIC algorithm achieves accuracy with compact hardware, its computation time is proportional to the number of bits used by the representation format.

Further author information:
António J. Araújo: aja@fe.up.pt

For a function composed of elementary functions, the CORDIC algorithm is slower, since it computes each elementary function sequentially. It is too slow for numerically intensive applications. Implementing a function by a single lookup-table is simple and very fast. For low precision computations, this kind of implementation is straightforward. For high precision computations, however, the single lookup-table implementation is impractical due to the huge table size that is required.

To reduce memory size, polynomial approximations have been used and extensive work exist.[4–8] This method approximates a given numerical function by piecewise polynomials, and realize the polynomials with hardware. For piecewise polynomial approximations, in many cases, the domain is partitioned into uniform segments. For elementary functions, such as $\sin x$ and e^x, by using higher-order polynomial approximations, the number of uniform segments can be reduced, and therefore the memory size can be reduced too. However, for some numerical functions, such as $\sqrt{-\ln x}$, methods based on uniform segmentation yield large memory size for implementation on conventional FPGAs even if second-order polynomials are used.[9] Polynomial-only or a combined table and polynomials solution, both reinforce the role of polynomial approximation.

The general method to calculate an elementary function is composed by three steps: range reduction, computation of the polynomial in the reduced argument and range reconstruction. This methodology allows a low-degree polynomial approximation but range reduction and reconstruction introduce a computation overhead. However, using such approach may be infeasible in high-demand applications as those that occurs in embedded systems and system-on-chip. In addition, data with specific formats are typical of such applications allowing to consider not doing range reduction, specially if low precision or a range with few bits is enough.

The main contribution of this work is a new approach to evaluate elementary functions opening a new paradigm to consider in design space exploration. It consists in performing evaluation without range reduction (woRR) and consequently without range reconstruction. The implementation of this methodology is compared to the common method consisting in evaluation with range reduction (wRR). The woRR method is used in a previous work,[10] but with other goal in mind. As far as we know, this is the first paper dealing with this issue.

The rest of the paper is organized as follows. Section 2 describes background material. Section 3 covers the proposed methodology to evaluate elementary functions without range reduction. Section 4 describes its implementation and related issues. Section 5 presents the experimental results and discusses them comparing the proposed method (woRR) with the general one (wRR). Main conclusions and future developments are described in section 6.

2. BACKGROUND

This section presents several aspects related to function evaluation. Firstly, the steps involved in the general method of elementar function evaluation are described. Then, some considerations are done about polynomial approximations.

2.1 Function evaluation

Consider an elementary function $f(x)$, where x have a given range $[a, b]$ and precision requirement. The evaluation of $f(x)$ typically consists of three steps:[11]

1. range reduction, reducing x over the interval $[a, b]$ to a more convenient x' over a smaller interval $[a', b']$,
2. function evaluation on the reduced interval, and
3. range reconstruction, expanding the result back to the original result range.

Range reduction allows to consider a small interval where approximations can be done with low degree polynomials. There are two main types of range reduction:

- additive reduction: $x' = x - kC$;
- multiplicative reduction: $x' = x/C^k$

where integer k and a constant C are specific for each elementary function, depending also of the approximation interval. The kind of reduction depends on the function to be evaluated.

Table 1 shows typical values of these parameters for several elementary functions, where MSB(x) refers to the most significant bit of x. These values are not universally accepted and for this reason other values can be find in the literature.

Table 1. Range reduction parameters for some functions.

Function	$\cos x$	e^x	$\log x$	$\sqrt{x}$
Reduction type	additive	additive	multiplicative	multiplicative
$[a', b']$	$[-\pi/4, \pi/4]$	$[0, \log 2]$	$[0.5, 1]$	$[0.25, 1]$
C	$\pi/2$	$\log(2)$	2	2
k	$\lfloor x/C \rceil$	$\lfloor x/C \rfloor$	MSB$(x)+1$	MSB$(x)+1$

2.2 Polynomial approximation

Polynomial approximation is the generic mathematical tool that reduces the evaluation of a function to additions and multiplications. A good primer on polynomial approximation for function evaluation is Mullers book.[11]

One may use the well-known Taylor or Chebyshev approximation polynomials of arbitrary degree.[11,12] These polynomials can be obtained analytically, or using computer algebra systems like Maple.[13] Another method of polynomial approximation is Remez's algorithm, a numerical process that, under some conditions, converges to the minimax approximation: the polynomial of degree n that minimizes the maximal difference between the polynomial $p(x)$ and the function $f(x)$ (equation 1).

$$\|f - p\|_\infty = \max_{a \le x \le b} |f(x) - p(x)| \tag{1}$$

Between approximation and evaluation, for an efficient hardware implementation, one has to round the coefficients of the minimax polynomial (which has real numbers in theory, and are computed with large precision in practice) to smaller precision numbers suitable for efficient evaluation. On a processor, one will typically try to round to single or double-precision numbers. On a FPGA, we may build adders and multipliers of arbitrary size, but the question is about the optimal size that these coefficients should have. This subject is discussed in a paper[14] where an error analysis is done considering separately the error of rounding each coefficient of the minimax polynomial and tries to minimize the bit-width of the rounded coefficients while remaining within acceptable error bounds.

To measure the numerical quality of a function evaluator, it should be considered the total error ϵ resulting from the implementation. This error has two components: approximation error ϵ_{ap} and quantization error ϵ_{qt} (equation 2).

$$\epsilon = \epsilon_{ap} + \epsilon_{qt} \tag{2}$$

The approximation error ϵ_{ap} is introduced in the function approximation step because it results from using polynomials to approximate a function. This error is independent of the used precision. The quantization error ϵ_{qt} is a consequence of the limited representation used do represent the polynomial coefficients. Moreover, it propagates between the operations realized during the value computation of a polynomial. This propagation is worsened when doing range reduction and reconstruction, present in the wRR general methodology. A trade-off should exists between these two parcels and the degree of an approximation.

3. PROPOSED METHODOLOGY

In spite of range reduction and range reconstruction allow approximations with low-degree polynomials, they introduce an additional computation overhead, since it involves operators like multipliers and dividers. This

reflects in the cost and latency of the hardware implementation. On average, range reduction and reconstruction can be as high as 37% on $\sin x$ and 23% on $\log x$.[14]

Custom computing hardware for embedded applications uses specific number formats. In most of these applications, range and precision are lesser than the used in corresponding software implementations, where standard floating point formats are used. Therefore, it these circumstances it is plausible doing function evaluation without range reduction. The success of that approach naturally depends of the range where the evaluation is needed and of the required precision.

A direct consequence of this methodology is the computation of higher degree polynomials comparing to the common three-step methodology. This is particularly true if only one polynomial is used to evaluate the function over the entire interval, what is not the most frequent case. To solve this issue we propose to use interval segmentation.[15] It consists in to slice the approximation interval into segments and compute the polynomial that approximates a specific segment, allowing function evaluation through low degree polynomials. Although this technique leads to more compact and efficient designs, it is more effective when applied to the proposed methodology since it only optimizes the polynomial computation, having no effect in range reduction and reconstruction steps of the general computing methodology.

Even so, polynomials tend to have higher degrees in the proposed method, which motivates a careful balance between the number of segments to choose and the degree of correspondent polynomials. This choice has a direct impact in terms of cost and latency of the implementations.

Regarding the proposed methodology and as far as we know, there is no published research about combining function evaluation without range reduction and interval segmentation.

4. DESIGN AND IMPLEMENTATION

This section details the main aspects concerning the design flow of the proposed methodology and also the adopted architecture for implementation.

4.1 Design flow

The design flow is depicted in figure 1, showing two main steps: numerical optimization and hardware implementation.

When need to design an evaluator for a certain function $f(x)$, the first things to define are the interval of x, $[a, b]$, where the evaluation is wanted, as well as the required precision for that evaluation. This is the start data to proceed to the numerical optimization, that was entirely performed with scripts for Maple and Matlab tools. Maple generates the coefficients of the polynomial $p(x)$ that approximate $f(x)$ and reports the approximation error ϵ. As it uses as entry the degree of the polynomial, this requires an iterative process, controlled by Matlab, in order to achieve the defined precision. In each iteration the degree is incremented by one and the approximation error is checked. This is done until the accuracy is achieved.

Regarding the segmentation, it is carried out when the estimated values of the latency and/or the area are above of reference values. These correspond to the obtained values for the entire set of range reduction, approximation and reconstruction, that are estimated by the logical synthesis tool. The number of segments is determined using the precision as criteria, conjugated with non-uniform segmentation that minimizes the number of slices that results for the whole interval. Moreover, it can adapts to the behaviour of a function in terms of linearity.

A Matlab script generates all the synthesizable Verilog code, that models the function evaluator. Logic synthesis of all models is done, reporting estimated values of cost implementation (area) and latency. Functional validation is done at that time. Besides polynomial extraction, Matlab was also used to furnish the results of a function evaluation to the simulation tool, Modelsim, to verify the correctness of the evaluation results, as well as the satisfiability of requirements for accuracy. After that, the implementation continues, doing place and route of the circuit. A bitstream file is created to programming the prototyping board, where the evaluator can be tested.

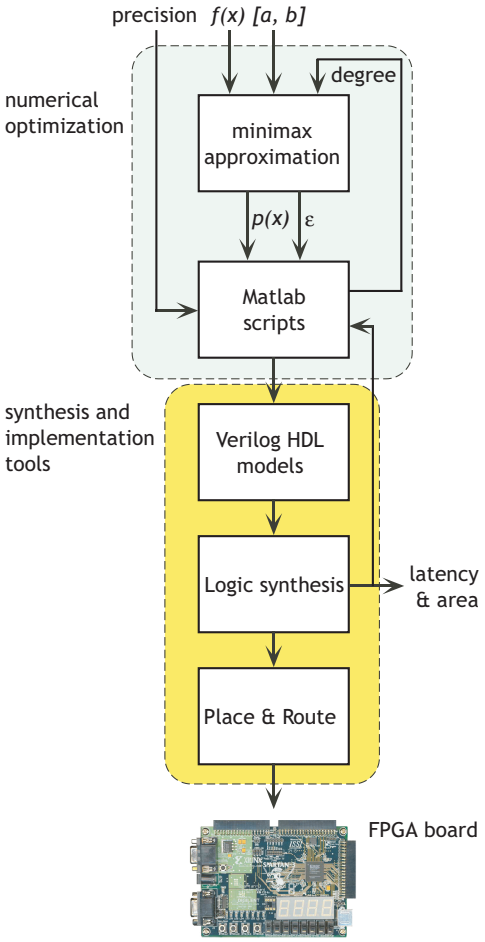


Figure 1. Implementation flow.

4.2 Architecture

The numerical optimization step produces a polynomial or a set of segments and a set of polynomials in the case of using segmentation. To represent these data, a fixed point representation is used. The range, fixed by the interval $[a, b]$, determines the number of bits of the integer part. The fractional part is conditioned by the required precision.

The general form of a n-degree polynomial (3) is not the best to implement it.

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0 \tag{3}$$

The total number of operations can be reduced using the Horner rule, that rewrites $p(x)$ as showed in equation 4.

$$p(x) = (_{\ldots}(c_n x + c_{n-1})x + \cdots)x + c_0 \tag{4}$$

Assuming the proposed woRR methodology, the architecture used to implement a function evaluator with segmentation is shown if figure 2.

The segment index block consists in a comparator that determines the segment corresponding to the input argument x. It determines the index of an entry on the table and forwards the computation to the correct polynomial that approximates the function in that segment. In case of a FPGA implementation, the table can be implemented using a BRAM (Block RAM). Additionally, also the FPGA embedded multipliers can advantageously be used.

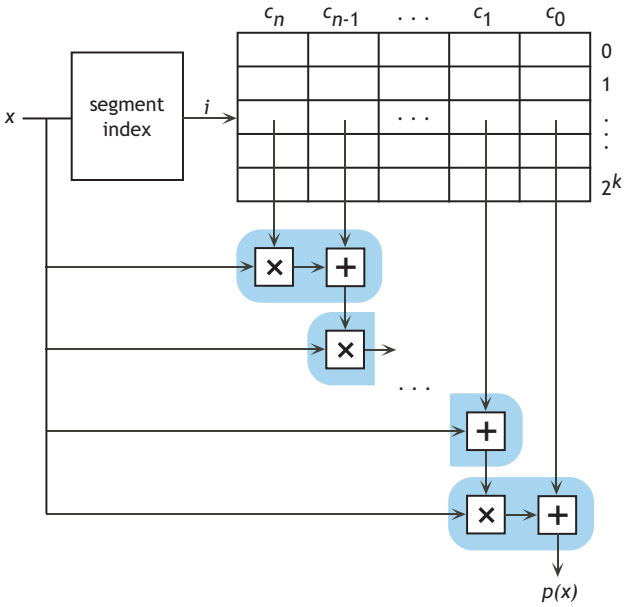


Figure 2. Architecture of an evaluator resulting of the woRR method with segmentation.

5. RESULTS

This section presents the experimental results that we obtained with the proposed approach using wRR and woRR methods, with and without interval segmentation. Results for wRR are reported since they are used as a reference to the ones of woRR. The elementary functions to which they relate are $\cos x$ and e^x, that are representative of the elementary functions class. Other ones were experimented but due to the amount of data produced, we restrict them to those two functions. Although, for two of these functions, $\log(x)$ and $\sqrt{x}$, was concluded that the woRR method reveals no advantage regarding both metrics, area and latency. In fact, they contain a high non-linearity near zero which forces high bit-width representation of the polynomial coefficients.

The results are provided for three possible values of precision using 8, 12 and 16 bits. The range considered for $\cos x$ is expressed by the value of the argument x. For that, three possible values are included: π, $3\pi/2$ and 2π. For e^x, range is also given by the argument value, considering in this case the values of 3, 4 and 5. These limited ranges are commonly found in elementary function evaluation for specific applications in embedded systems.[16,17]

The results are expressed in terms of the implementation cost, given by the implementation area, i.e., FPGA look-up tables (LUTs), and latency. The Xilinx ISE 12.1 toolkit was used to synthesize and implement the several circuits. The test was done in a prototyping board[18] equipped with a Spartan 3 FPGA.

5.1 Implementations without segmentation

The results included in this section do not consider interval segmentation.

Table 2 shows the polynomial degree needed to evaluate $\cos x$ for both wRR and woRR methods. With the 16 bits precision it is not considered the range of 2π because the results are further out of the other values, which makes comparison with other results difficult. Table 3 shows the polynomial degree needed to evaluate e^x for both wRR and woRR methods. With the 16 bits precision it is not considered the range equal to 5 by the same reason given for $\cos x$.

Figure 3 presents the implementation results for $\cos x$, expressed by the area and latency metrics. For each range, two sets of bars are given: one for wRR and other one for woRR method. Each set of bars refers to the considered precision values. Comparing the results for the same precision, the main conclusion is that area is

Table 2. Polynomial degree used to approximate $\cos x$ without segmentation.

Precision	wRR			woRR		
	π	$3\pi/2$	2π	π	$3\pi/2$	2π
8 bits	3	3	3	5	5	6
12 bits	4	4	4	5	6	7
16 bits	5	5	-	7	8	-

Table 3. Polynomial degree used to approximate e^x without segmentation.

Precision	wRR			woRR		
	3	4	5	3	4	5
8 bits	3	3	4	5	6	8
12 bits	4	4	5	6	8	9
16 bits	5	5	-	7	9	-

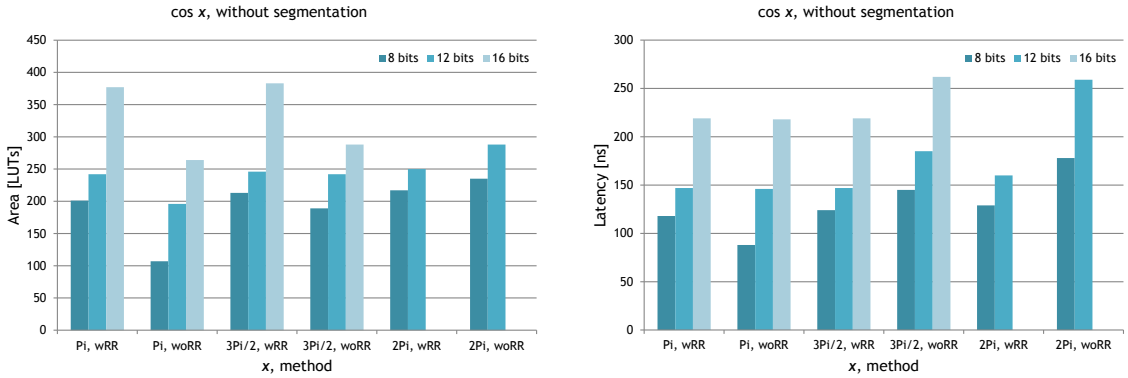
slightly lower in woRR method, but only for ranges up $3\pi/2$. Latency increases from $3\pi/2$. This suggests that interval segmentation can be useful to improve woRR results, at least for one of the metrics.

The implementation results for e^x are presented in figure 4. In all the considered ranges, it is visible a better result with regard to the area. This is explained by the need of performing range reduction and range reconstruction in wRR method, in spite of in woRR proposed method the polynomials have a greater degree than the used for wRR.

5.2 Implementations with segmentation

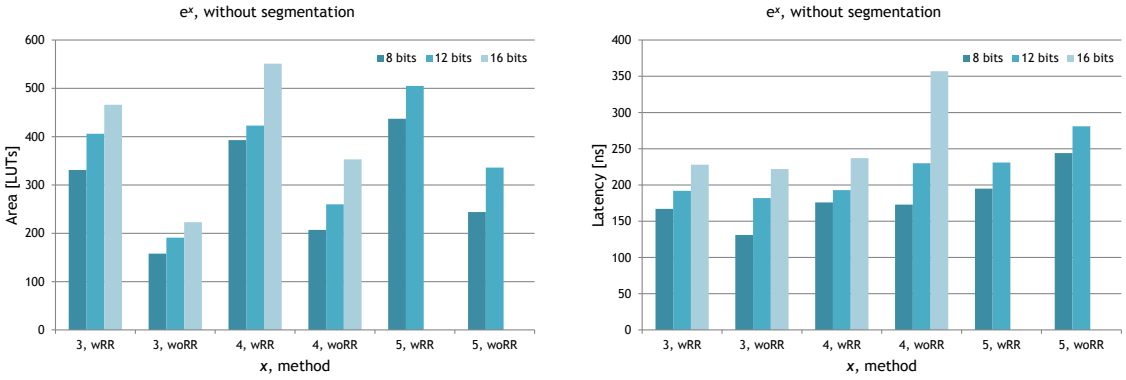
The results included in this section consider now interval segmentation.

The consequence of using interval segmentation is more visible with the woRR design methodology we proposed as previously seen. For example, table 4 shows the number of clock cycles needed to perform the evaluation of $\cos x$ for both, wRR and woRR. It confirms that, for the same range and precision, the polynomial degree for woRR is lesser than the one for wRR. Of course this conclusion is valid only for the considered range and precision.



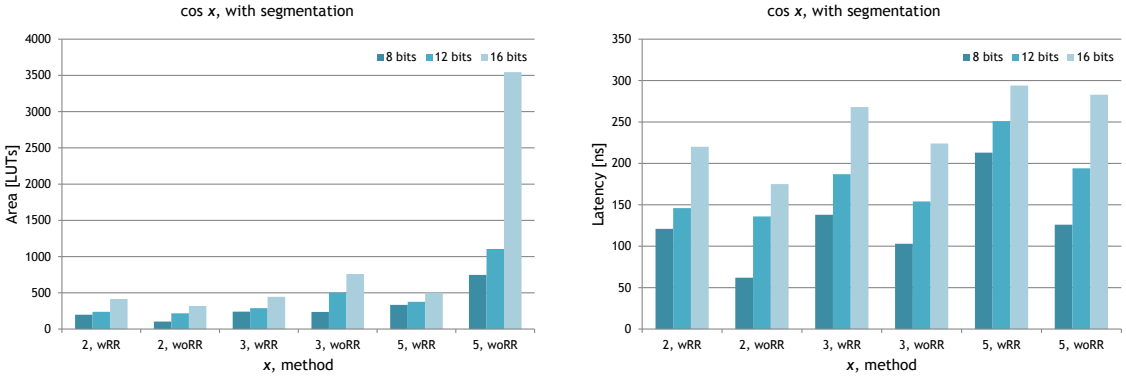
(a) (b)

Figure 3. Results of the evaluation of $\cos x$ without interval segmentation: area (a) and latency (b).



(a) (b)

Figure 4. Results of the evaluation of e^x without interval segmentation: area (a) and latency (b).

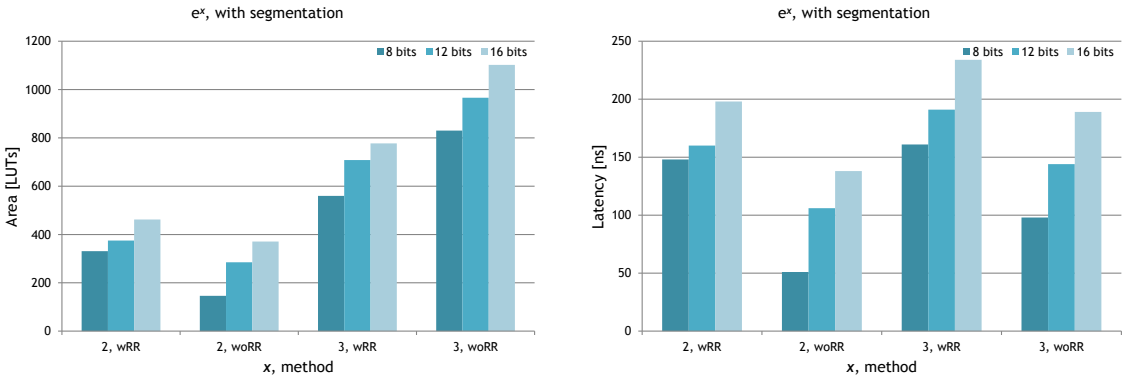


(a) (b)

Figure 5. Results of the evaluation of $\cos x$ with interval segmentation: area (a) and latency (b).

Table 4. Clock cycles to evaluate $\cos x$ with segmentation.

Precision	wRR				woRR			
	2	3	4	5	2	3	4	5
8 bits	12	13	14	15	6	6	6	6
12 bits	14	15	16	17	8	8	8	8
16 bits	16	17	18	19	10	10	10	10



(a) (b)

Figure 6. Results of the evaluation of e^x with interval segmentation: area (a) and latency (b).

The results for $\cos x$ and e^x are shown in figures 5 and 6, respectively.

Analysing these figures, is interesting to note that latency is always reduced with the methodology we proposed (woRR) face to the general methodology (wRR). However, the same not happens with area. Except for low ranges, the area increases when using the woRR method. This is due to the size of table used to save the coefficients of polynomials for each segment and the correspondent index computation block.

5.3 Analysis

The main results obtained for computation latency are summarized in Table 5. As it can be seen from the table, it is advantageous to avoid range reduction for the functions $\cos x$ and e^x, when computing them over the covered precisions and ranges. This is mainly due to the use of a divider to perform range reduction in both functions, which greatly increases the computation time.

Table 5. Latency reduction without range reduction and using interval segmentation.

Precision	$\cos x$				e^x	
	2 bits	3 bits	4 bits	5 bits	2 bits	3 bits
8 bits	50%	28%	26%	42%	66%	40%
12 bits	8%	11%	21%	22%	36%	24%
16 bits	21%	18%	14%	4%	29%	16%

Regarding the occupied area, this method only optimizes ranges up to 2 bits, and for a particular case of 3 bits for $\cos x$ with a precision of 8 bits. When computing $\cos x$ for ranges higher than 5 bits, the cost tends to be very high. Computing e^x for ranges higher than 3 bits reveals no advantage at all in giving up range reduction

because of the high non-linearity behaviour of this function resulting the same issue as for $\log x$ and $\sqrt{x}$ near zero.

Comparing these implementation results to others from previous works is difficult, specially because there are no results regarding evaluation without range reduction together with segmentation.

6. CONCLUSION

The general methodology to evaluate elementary functions in any point of their domains is formed by three steps. Although this procedure is generally considered advantageous for most situations, in some specific applications typically found in embedded systems, elementary functions need to be computed in small intervals of their domains. This circumstances combined with certain accuracy requirements, they do not require high degree polynomials and therefore may improve the computation without range reduction and reconstruction steps. The methodology proposed in this work explores these occurrences and proposes to use interval segmentation allied to evaluation without range reduction. The methodology was applied for some elementary functions, that were evaluated in a FPGA platform, confirming the expectations. Experimental results show a significant reduction of latency for $\cos x$ and e^x evaluators, depending of the required precision of computation.

As future improvements, it is expected to automate the procedures related to approximation optimization and generation of synthesizable models. This is considered fundamental to improve the design space exploration.

ACKNOWLEDGMENTS

The authors wish to acknowledge the support of FCT (Fundação para a Ciência e a Tecnologia), under the Associate Laboratory contract with INESC Porto.

REFERENCES

[1] O'Grady, E. P. and Wang, C.-H., "Performance limitations in parallel processor simulations," *Transactions of the Society for Computer Simulation International* **4**, 311–330 (October 1987).

[2] Volder, J., "The CORDIC computing technique," *IRE Transactions on Electronic Computers* **8**, 330–334 (1959).

[3] Andraka, R., "A survey of CORDIC algorithms for FPGA based computers," in [*Proceedings of the 6th. International Symposium on Field Programmable Gate Arrays*], 191–200 (February 1998).

[4] Cao, J., Wei, B. W. Y., and Cheng, J., "High-performance architectures for elementary function generation," in [*Proceedings of the 15th IEEE Symposium on Computer Arithmetic*], 136–144 (June 2001).

[5] Defour, D., de Dinechin, F., and Muller, J.-M., "A new scheme for table-based evaluation of functions," in [*Proceedings of the 36th Asilomar Conference on Signals, Systems and Computers*], 1608–1612 (November 2002).

[6] Pineiro, J.-A., Oberman, S. F., Muller, J.-M., and Bruguera, J. D., "High-speed function approximation using a minimax quadratic interpolator," *IEEE Transactions on Computers* **54**, 304–318 (March 2005).

[7] Lee, D.-U., Cheung, R. C. C., Luk, W., and Villasenor, J. D., "Hardware implementation trade-offs of polynomial approximations and interpolations," *IEEE Transactions on Computers* **57**, 686–701 (May 2008).

[8] de Dinechin, F., Joldes, M., and Pasca, B., "Automatic generation of polynomial-based hardware architectures for function evaluation," in [*Proceedings of the 21st IEEE International Conference on Application-Specific Systems, Architectures and Processors*], 216–222 (July 2010).

[9] Nagayama, S., Sasao, T., and Butler, J. T., "Design method for numerical function generators based on polynomial approximation for FPGA implementation," in [*Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*], 280–287 (August 2007).

[10] Lee, D.-U., Gaffar, A. A., Mencer, O., and Luk, W., "Adaptive range reduction for hardware function evaluation," in [*Proceedings of the IEEE International Conference on Field-Programmable Technology*], 169–176 (December 2004).

[11] Muller, J.-M., [*Elementary Functions – Algorithms and Implementation*], Birkhauser, Boston, 2nd ed. (2006).

[12] Araújo, A. J. and Matos, J. S., "Approximation of elementary functions by polynomials and rational functions - some results for FPGA based implementations," in [*Proceedings of the 11th Symposium on Computer Architecture and High Performance Computing*], 257–264 (September 1999).

[13] Char, B. W., Geddes, K. O., Gonnet, G. H., Leong, B. L., Monagan, M. B., and Watt, S. M., [*Maple V Library Reference Manual*], Springer Verlag, Berlin, Germany (1991).

[14] Lee, D.-U., Gaffar, A. A., Mencer, O., and Luk, W., "Optimizing hardware function evaluation," *IEEE Transactions on Computers* **54**, 1520–1531 (December 2005).

[15] Detrey, J. and de Dinechin, F., "Table-based polynomials for fast hardware function evaluation," in [*Proceedings of the 16th IEEE International Conference on Application-Specific Systems, Architectures and Processors*], 328–333 (July 2005).

[16] Cheung, R. C. C., Lee, D.-U., Mencer, O., Luk, W., and Cheung, P. Y. K., "Automating custom-precision function evaluation for embedded processors," in [*Proceedings of the ACM/IEEE International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*], 22–31 (September 2005).

[17] Lee, D.-U. and Villasenor, J. D., "Optimized custom precision function evaluation for embedded processors," *IEEE Transactions on Computers* **58**, 46–59 (January 2009).

[18] Xilinx, *Spartan–3 Starter kit board User Guide* (2005). `http://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD_RM.pdf`.