Approximation of Elementary Functions by Polynomials and Rational Functions – Some Results for FPGA Based Implementations

António J. Araújo, José S. Matos

Faculdade de Engenharia da Universidade do Porto
Instituto de Engenharia de Sistemas e de Computadores
Pr. da República, 93 - Apartado 4433 - 4007 Porto CODEX - Portugal
{aaraujo,jsm}@picasso.inescn.pt

Abstract—

The hardware evaluation of elementary functions such as $exp(x)$ and $sin(x)$ is advantageous due to the performance that can be reached when compared with software solutions. Such evaluations can be done by computing simple functions that approximate the targeted elementary functions. Actual microprocessors and specific processors like the digital signal processors include computing capabilities for some elementary functions but they could be inappropriate for some situations. Due to the great impact that Field Programmable Gate Arrays (FPGA) have in a wide range of hardware applications, specially in areas where hardware customization can improve the overall performance, the implementation of these functions in FPGA devices is worth considering. The reconfigurability they offer increases the flexibility and the power of such custom computing machines. Implementations of common elementary functions using single precision floating point arithmetic are described and the correspondent results are presented.

Keywords— **Elementary functions, floating point arithmetic, FPGA, polynomial approximations**

I. Introduction

One of the techniques used to compute the value of a particular mathematical function is based on polynomial approximations [Mul97], as their evaluation requires a finite number of additions, subtractions and multiplications. If division is also available, then rational functions can be evaluated too. Polynomials and rational functions are then good choices to approximate elementary functions such as $e^x, logx, sinx$ and $arctgx$.

FPGAs have become a competitive alternative for high performance applications, like DSP, previously dominated by general purpose microprocessors and ASIC devices. An important feature when using them, refers to the possibility they offer that allows the development of a particular hardware application, optimized for different purposes under speed/area trade-offs. Another important advantage from using FPGAs is reconfigurability which can be explored to implement different approximations to a function using as criteria the allowable error. This important feature is not possible in current processors, where computing time is the same regardless of required precision. Customized applications can use FPGAs as a co-processor of a host as well as a stand-alone computing engine. The hardware implementations that will be described were developed with Viewlogic and Foundation 1.5 software packages and the Xilinx XC4000E family FPGAs [Xil96] were used.

Sections II and III introduce some aspects of approximating elementary functions by polynomial and rational functions. Some of the most common approximant polynomials are described. Section IV presents an example of how to obtain these approximations. Section V discusses the architectural alternatives to implement them and describes the floating point arithmetic operators used. Next, in section VI, the approximations for several functions are characterized. Their hardware implementations are described and the results are discussed. Finally, section VII highlights key aspects of the current work and refers future enhancements and developments.

II. Polynomial Approximations

Weierstrass's theorem guarantes that any continuous function can be approximated by a polynomial on an interval $[a, b]$ of its domain.

Let be f the function to be evaluated and $\mathcal{P}$ the set of polynomials with degree less or equal to n. Two kinds of approximations are generally considered: the approximations that minimize the average error, called *least squares approximations*, and the approximations that minimize the worst case error, called *least maximum approximations*, also known as *minimax approximations*. In both cases, the problem is to find a polynomial $p^* \in \mathcal{P}$ that minimizes the distance to f.

A. Least Squares Polynomial Approximations

Let be $p^*(x) = p_n^* x^n + p_{n-1}^* x^{n-1} + \cdots + p_1^* x + p_0^*$ the polynomial that minimizes equation 1, where w is a weight function that can be used to select parts of $[a, b]$ where the approximation should be more accurate.

$$\|f-p\| = \sqrt{\int_a^b w(x)[f(x)-p(x)]^2 dx} \quad (1)$$

Consider $\langle f, g\rangle$ the inner product defined by equation 2, that allows to recognize where two functions are ortoghonal.

$$\langle f, g\rangle = \int_a^b w(x)f(x)g(x)dx \quad (2)$$

The polynomial p^* can be computed as follows:

- build a sequence (T_m), $(m \leq n)$ of polynomials such that (T_m) is of degree-m and $\langle T_i, T_j\rangle = 0$ for $i \neq j$ (orthogonal polynomials);
- compute the intermediate coefficients

$$a_i = \frac{\langle f, T_i\rangle}{\langle T_i, T_i\rangle}; \quad (3)$$

- and finally compute

$$p^* = \sum_{i=0}^{n} a_i T_i. \quad (4)$$

Some sequences of orthogonal polynomials are well known and will be presented next. All of the approximations are on the interval $[-1, 1]$. However getting an approximation for another interval $[a, b]$ is straightforward:

- let $g(u) = f(x)$ where $u \in [-1, 1]$ and $x \in [a, b]$, with $x = \frac{b-a}{2}u + \frac{a+b}{2}$;
- compute a least squares approximation q^* to g in $[-1, 1]$;
- obtain the least squares approximation p^* to f as $p^*(x) = q^*(\frac{2}{b-a}x - \frac{a+b}{b-a})$.

A.1 Legendre Polynomials

For these polynomials, $w(x) = 1$ is considered and $[a, b] = [-1, 1]$.

$$\begin{cases} T_0(x) = 1 \\ T_1(x) = x \\ T_n(x) = \frac{2n-1}{n}xT_{n-1}(x) - \frac{n-1}{n}xT_{n-2}(x); \end{cases} \quad (5)$$

and

$$\langle T_i, T_j\rangle = \begin{cases} 0 & \text{if } i \neq j \\ \frac{2}{2i+1} & \text{otherwise.} \end{cases} \quad (6)$$

A.2 Chebyshev Polynomials

The weight function is $w(x) = 1/\sqrt{1-x^2}$ and $[a, b] = [-1, 1]$.

$$\begin{cases} T_0(x) = 1 \\ T_1(x) = x \\ T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) = \cos(n \arccos x); \end{cases} \quad (7)$$

and

$$\langle T_i, T_j\rangle = \begin{cases} 0 & \text{if } i \neq j \\ \pi & \text{if } i = j = 0 \\ \frac{\pi}{2} & \text{otherwise.} \end{cases} \quad (8)$$

A.3 Jacobi Polynomials

In such case $w(x) = (1-x)^\alpha(1+x)^\beta$ with $\alpha, \beta > 1$ and $[a, b] = [-1, 1]$.

$$T_n(x) = \frac{1}{2^n}\sum_{m=0}^{n}\binom{n+\alpha}{m}\binom{n+\beta}{n-m}(x-1)^{n-m}(x+1)^m. \quad (9)$$

B. Least Maximum Polynomial Approximations

Considering the distance between a function f and a degree-n polynomial p on a closed interval $[a, b]$ (equation 10), the polynomial p^* that satisfies equation 11 is a minimax degree-n polynomial approximation to f on $[a, b]$.

$$\|f-p\| = \max_{a \leq x \leq b} |f(x)-p(x)| \quad (10)$$

$$\|f-p^*\| = \min_{p \in \mathcal{P}} \|f(x)-p(x)\| \quad (11)$$

A theorem, due to Chebyshev, characterizes the minimax based approximants to a function and is used by an algorithm that computes the minimax polynomial approximation [HCL+68, Ric64].

III. Rational Approximations

Another theorem due to Chebyshev characterizes a minimax based rational function obtained by dividing a degree-n polynomial by another one of degree-m. Also an algorithm exists to generate such approximations [HCL+68]. Another solution for getting these approximations is by Padé approximants [Bak75], but a serious drawback exists since they only give local approximations.

It is difficult to say which kind of approximations are better for a given elementary function. Functions with a highly nonpolynomial behavior (for example, with finite limits at $\pm\infty$ or infinite derivatives) are better approximated by rational functions than by polynomials [Mul97]. However, the approximation by rational functions requires the division operator available in hardware. Although a rational approximation requires only one division, this operation is currently

performed by instructions that have a large latency even on actual microprocessors [Mul97].

IV. An Example

All the computation requirements to approximate elementary functions by the presented methods can actually be done by scientific applications such as Maple [CGG+91] avoiding a manual hard work. To see how Maple's package for numerical approximations, *numapprox*, can be used for this purpose, an example is included. The goal is to approximate the function $f(x) = e^x$ by a degree-3 Chebyshev polynomial. The main commands in the correspondent Maple program are self explanatory. Both Maple source code and obtained results are included.

```
> restart;
> with(numapprox):
> with(orthopoly):
> with(plots):
> setoptions(axes=boxed, axes-
font=[TIMES,ROMAN,10],
> titlefont=[TIMES,BOLDITALIC,10]);
> f := exp(x); Digits := 10:
> a:=-1:  b:=1:  n:=3:
> # Chebyshev
> w:=(1-x^2)^(-1/2):
> coef := array(0..n):
> poly := array(0..n):
> printf("Chebyshev based approximation
with n=%d\n", n);
> plot(w, x=a..b, title="w(x)");
> for i from 0 to n do
> poly[i] := T(i,x);
> coef[i] := evalf(int(w*f*poly[i],
x=a..b) / int(w*poly[i]^2,
> x=a..b));
> od;
> approx := sum(coef[k]*poly[k],
k=0..n):
> p(x) := collect(approx,x);
> plot({f, approx}, x=a..b, title="f(x)
and p(x)");
> ferror := f-approx;
> plot(ferror, x=a..b, title="error");
> maxerror := infnorm(ferror, x=a..b,
'xmax');
> x := xmax;
```

$$f := e^x$$

```
Chebyshev based approximation with n=3
```

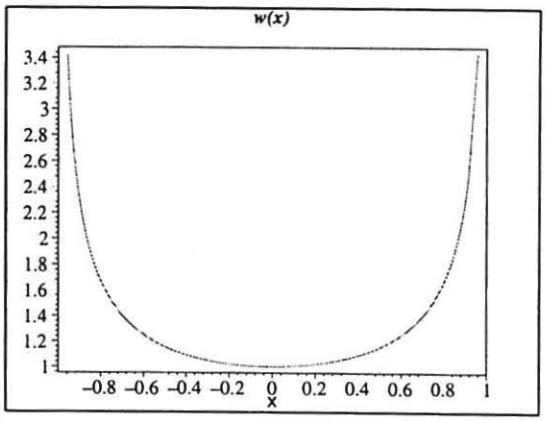

Fig. 1. Weight function.

$$poly_0 := 1$$

$$coef_0 := 1.266065878$$

$$poly_1 := x$$

$$coef_1 := 1.130318208$$

$$poly_2 := 2x^2 - 1$$

$$coef_2 := .2714953396$$

$$poly_3 := 4x^3 - 3x$$

$$coef_3 := .04433684984$$

$$p(x) := .9945705384 + .9973076585\,x + .5429906792\,x^2 + .1773473994\,x^3$$

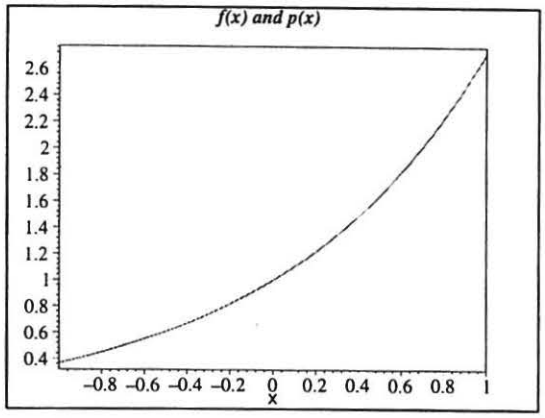

Fig. 2. Functions $f(x)$ and $p(x)$.

$$ferror := e^x - .9945705384 - .9973076585\,x - .5429906792\,x^2 - .1773473994\,x^3$$

$$maxerror := .006065552959$$

$$x := 1.$$

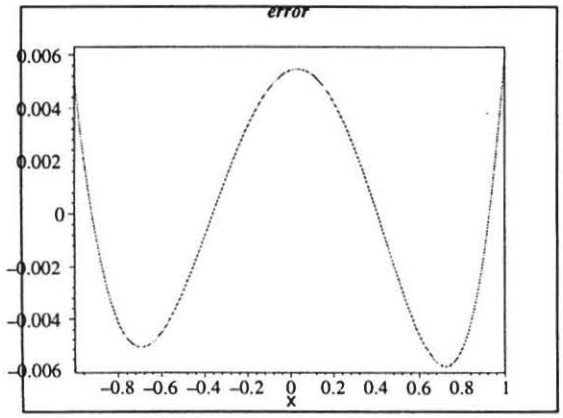

Fig. 3. Error function.

All the computations were performed with an accuracy of 10 digits, but other values could be specified. Figure 1 shows the weight function used (see section .0), chosen in a way as to put more precision on the interval extremes. From figure 2 the curves of $f(x)$ and $p^*(x)$ are indistinctive on that scale, due to the small approximation error. Figure 3 shows the error curve and it can be shown at the end of the results list that $p^*(x)$ exhibits the maximum error (0.006) at $x = 1$ when approximating e^x.

V. Hardware Implementation

The main hardware resources needed for polynomial evaluation are an adder/subtracter and a multiplier, and also a divider for rational functions, all of them for floating point operands. Next sections detail these requirements.

A. Polynomial Evaluation

In order to minimize the number of arithmetic operations to evaluate a general polynomial Horner's rule can be used. With this scheme a polynomial like $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ can be computed as $(((a_4x + a_3)x + a_2)x + a_1)x + a_0$. By this way, the power operator is avoided and the total number of performed operations is reduced. For a complete degree-n polynomial, with Horner's rule, only n multiplications and n additions must be executed.

For large degree polynomials another method called adaptation of coefficients [Knu81] can be used, and other methods exist that take in account the number of nonexisting polynomial terms [Mul97].

Figure 4 shows an arranjement that can be used to evaluate polynomials with a topology that follows Horner's rule. For a degree-n polynomial at most $2n$ clock cycles are needed.

For rational function computation two alternatives exist. An obvious solution uses an evaluator for the numerator and another one for the denominator (figure 5(a)). The division

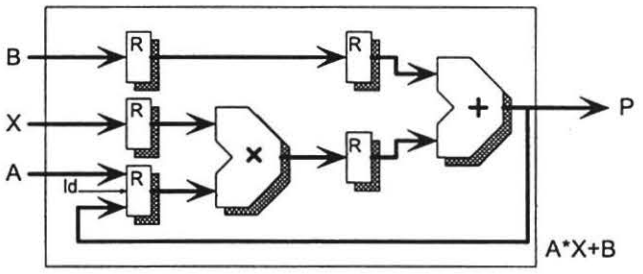

Fig. 4. Polynomial evaluation.

is only enabled when both polynomials evaluation are completed and the final result is available in $2m + 1$ clock cycles. The other way a rational function can be computed uses only a polynomial evaluator (figure 5(b)) that performs both numerator and denominator evaluations in $2(n + m) + 1$ clock cycles. Appropriated control enables the correct load operation into the register. This solution consumes less area but is slower than the first one.

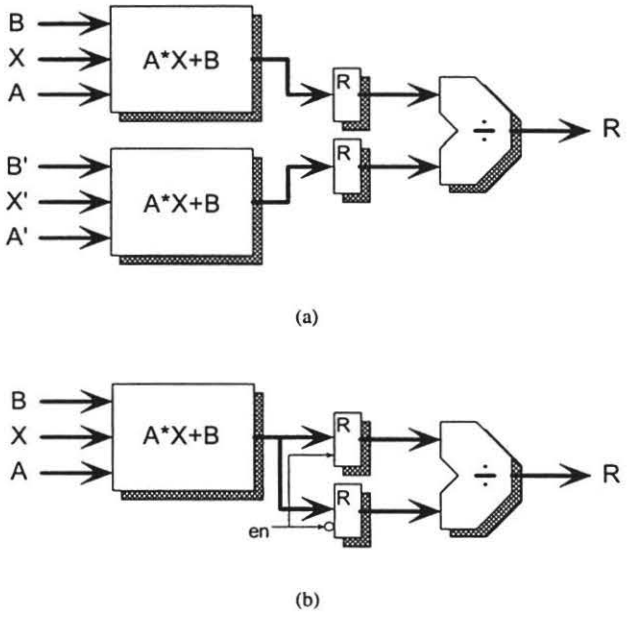

Fig. 5. Alternatives for rational function evaluation.

B. Floating Point Operators

The three arithmetic operators needed to perform polynomial and rational function evaluations were designed to execute single precision floating point operations [Boa85]. This format uses a 23-bit wide significand, a 8-bit wide exponent and 1 bit to store the operand's signal.

The main operation when performing floating point computations uses integer arithmetic to operate the significands and also the exponents of each operand. Due to the number of bits, the integer operations involving the significands are the most time consuming tasks in the floating point algorithms for the addition/subtraction, multiplication and division. In the implementations done, these integer operations were implemented as parallel arrays to optimize the speed. A common operator that is present in all their cells is a full adder. The Xilinx XC4000E family [Xil96] provides inside each configurable logic block (CLB) dedicated logic for carry generation and exclusive paths for fast carry propagation between adjacent CLBs [New96]. This dedicated carry logic allows to implement two full adders inside each CLB. This signifies that an integer adder with 24 bits needs only 12+2 CLBs. The two additional CLBs are needed to initialize and terminate the carry chain [New96].

The integer units used by the operators showned at figures 4, 5(a) and 5(b) are non-pipelined. However to evaluate large degree polynomials or when evaluating a stream of polynomials, even if each one is a low degree polynomial, a pipelined solution could be preferable in order to increase the thoughput of the evaluator.

TABLE I
IMPLEMENTATION DATA FOR THE OPERATORS

Operator	*CLBs*	*Delay time (ns)*
Adder/Subtracter	146	72
Multiplier	682	105
Divider	428	310

Table I shows the results for the three floating point operators that were implemented, showing for each one the number of CLBs and the delay time that limits the performance. The largest unit is the multiplier, consuming 682 CLBs. This implies that at least a X4025 FPGA should be used in a circuit that uses such multiplier. In spite of the integer divider size be smaller than the integer multiplier, the divider's delay time is greater than the multiplier's delay time because all the cells of the array are in the carry path [Kor98].

VI. RESULTS

To obtain an elementary function approximation a two step procedure is used. First, Maple is used to obtain the approximant and then it is evaluated with a choosed architecture. Many approximations were obtained for several elementary functions in $[-1, 1]$, using the described approximation methods (sections II and III). Some of them are included here, focusing on the approximation error or the number of significant bits.

Tables II and III present the maximum absolute errors for e^x and $ln(x+2)$ when using several approximants with degrees from 2 to 5.

TABLE II
MAXIMUM ABSOLUTE ERRORS FOR e^x

Degree	*Legendre*	*Chebyshev*	*Jacobi*	*minimax*
2	8.2×10^{-2}	5.0×10^{-2}	1.2×10^{-1}	4.5×10^{-2}
3	1.1×10^{-2}	6.1×10^{-3}	1.9×10^{-2}	5.5×10^{-3}
4	1.2×10^{-3}	5.9×10^{-4}	2.3×10^{-3}	5.5×10^{-4}
5	1.1×10^{-4}	4.8×10^{-5}	2.3×10^{-4}	4.5×10^{-5}

TABLE III
MAXIMUM ABSOLUTE ERRORS FOR $ln(x+2)$

Degree	*Legendre*	*Chebyshev*	*Jacobi*	*minimax*
2	2.6×10^{-2}	1.6×10^{-2}	3.8×10^{-2}	1.3×10^{-2}
3	6.0×10^{-3}	3.3×10^{-3}	1.0×10^{-2}	2.7×10^{-3}
4	1.4×10^{-3}	7.1×10^{-4}	2.7×10^{-3}	5.8×10^{-4}
5	3.5×10^{-4}	1.6×10^{-4}	7.5×10^{-4}	1.3×10^{-4}

From these results it is clear that the minimax method is the best, followed by Chebyshev's approximants. These functions were analyzed but other ones could be used instead. Figure 6 visualizes the error curves correspondent to the approximations of e^x by degree-3 polynomials. From these plots can be observed that Jacobi polynomials present bad results due to its behaviour near the interval extremes.

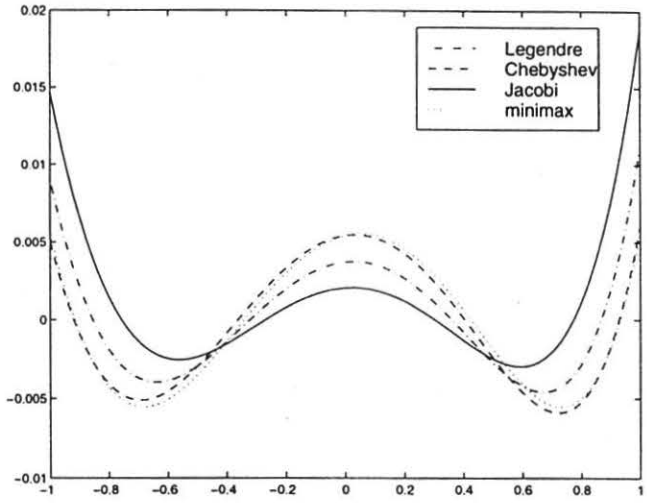

Fig. 6. Error curves for degree-3 approximants.

Applying the minimax method on different functions and for several degrees, the significant bits of the results are on the table IV. The number of significant bits is computed from the maximum absolute error by equation 12.

$$\mathrm{NSB} = -\log_2 |f(x) - p^*(x)| \qquad (12)$$

TABLE IV
NSB WHEN APPROXIMATING SEVERAL FUNCTIONS BY MINIMAX

$f(x)\backslash n$	2	3	4	5	6	7	8
e^x	4.5	7.5	10.8	14.4	18.2	22.3	26.4
$\ln(x+2)$	6.2	8.5	10.7	12.9	15.0	17.1	19.2
$\sin x$	4.7	11.0	11.0	18.3	18.3	26.5	26.5
$\arcsin x$	2.4	3.5	3.5	4.2	4.2	4.7	4.7
$\tan x$	2.7	5.6	5.6	8.6	8.6	11.5	11.5

A conclusion that can be taken from table IV is that the sequence of minimax polynomials converges to $f(x)$ with different speeds and that the convergence speed is difficult to predict. For example, the approximations to the function *arcsin(x)* present bad results. Even with large degree polynomials its approximations error decreases slowly. Figure 7 is an alternative to table IV and helps to see these situations.

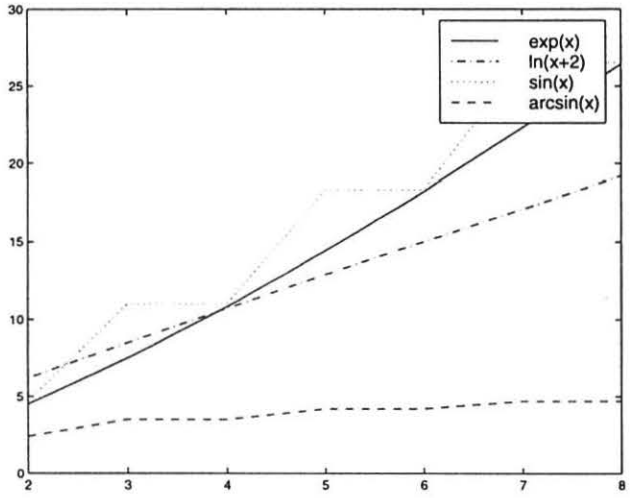

Fig. 7. Evolution of the NSB using minimax.

For several elementary functions, figure 8 shows de minimum polynomial degree that is needed to reach a target accuracy on the result. For these results Chebyshev polynomials were used. The error curves for *arcsin(x)* correspondent to several degree-n polynomials are presented in figure 9.

Results from using rational function approximations are also included. Table V includes the absolute maximum error and the NSB for both polynomial and rational functions when approximating e^x in $[-1, 1]$, varying n from 2 to 5. At the first column $(n, 0)$ designates a degree-n polynomial and (n, m) refers to the rational function degrees used for the numerator and denominator polynomials.

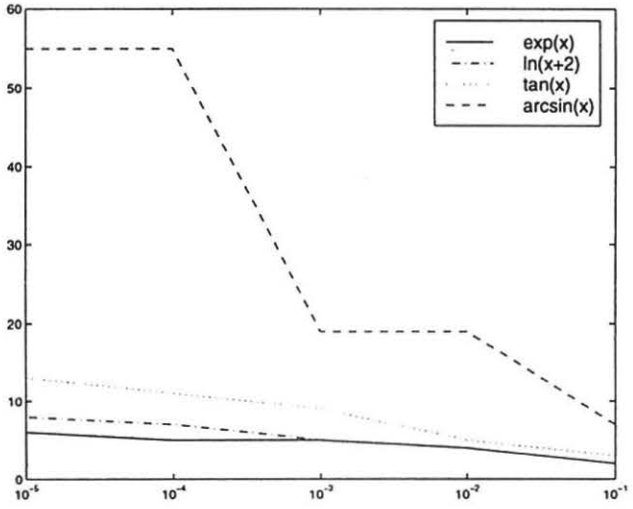

Fig. 8. Polynomial's degree as a function of accuracy.

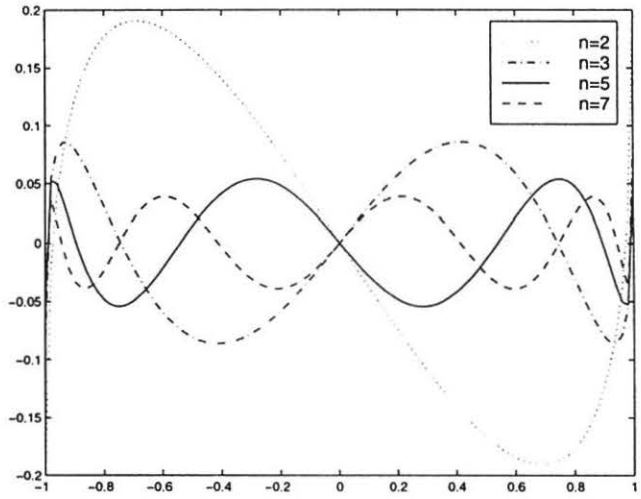

Fig. 9. Error functions for $arcsin(x)$.

An important feature can be taken from these data. In terms of NSB the table entries for $(n, m) = (2, 3)$ and $(n, m) = (5, 0)$ show that the first one presents a better result than the second one. This better precision is accomplished by a faster evaluation using the circuit on figure 5(a) where two polynomials are calculated in parallel. The rational approximation for $(n, m) = (2, 3)$ takes 7 clock cycles while the polynomial approximation consumes 10 clock cycles with $(n, m) = (5, 0)$. These results were obtained for e^x, but the same conclusion can be obtained from other elementary functions.

Table VI quantifies the clock cycles needed to compute an

TABLE V
RESULTS FOR e^x RATIONAL APPROXIMATIONS

n,m	|error|	NSB
2,0	4.5×10^{-2}	4.5
0,2	3.5×10^{-2}	4.8
1,2	1.7×10^{-3}	9.2
2,2	8.7×10^{-5}	13.5
3,0	5.5×10^{-3}	7.5
0,3	4.5×10^{-3}	7.8
1,3	1.2×10^{-4}	13.0
2,3	4.3×10^{-6}	17.8
3,3	1.6×10^{-7}	22.6
4,0	5.5×10^{-4}	10.8
0,4	4.6×10^{-4}	11.1
1,4	8.2×10^{-6}	16.9
2,4	2.0×10^{-7}	22.2
3,4	5.5×10^{-9}	27.4
4,4	1.5×10^{-10}	32.6
5,0	4.5×10^{-5}	14.4
0,5	3.9×10^{-5}	14.6
1,5	4.9×10^{-7}	21.0
2,5	9.0×10^{-9}	26.7
3,5	1.9×10^{-10}	32.3
4,5	4.2×10^{-12}	37.8
5,5	9.7×10^{-14}	43.2

exponential in $[-1, 1]$, using different degrees for minimax polynomials and rational functions. Column 2 refers to the polynomial evaluator of figure 4, and columns 3 and 4 refer to the rational function evaluators presented in figures 5(a) and 5(b), respectively.

The curves showned at figure 10 give a better understand about this behavior. The approximation accuracy in terms of NSB is presented as a function of the number of clock cycles, both for polynomial and rational approximations of e^x with minimax method. From this data can be concluded that in terms of hardware it is possible to obtain a solution that is simultaneously faster and precise than other ones. For example, while a degree-5 polynomial takes 10 clock cycles to evaluate e^x, this function can be also computed in only 7 clock cycles by the parallel evaluator of figure 5(a) with increased precision. This is the main conclusion that can be taken from this analysis. However, the area needed for these implementation alternatives and consequently the FPGA prices with such resources should be taken into account.

VII. CONCLUSION

The main contribution of this work is on the analysis of feasibility of elementary function approximations for FP-

TABLE VI
CLOCK CYCLES FOR e^x APPROXIMATIONS

n,m	P	R type 1	R type 2
3,0	6		
0,3		7	7
1,3		7	9
2,3		7	11
3,3		7	13
5,0	10		
0,5		11	11
1,5		11	13
2,5		11	15
3,5		11	17
4,5		11	19
5,5		11	21

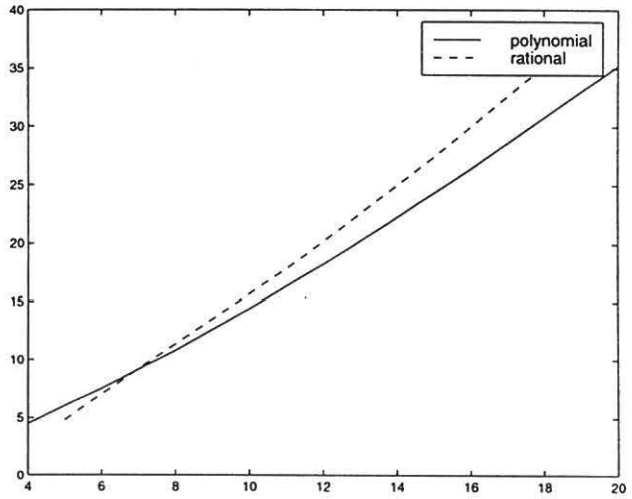

Fig. 10. NSB as a function of clock cycles.

GAs based custom computing machines. Some results were shown for an implementation that uses non-pipelined floating point arithmetic operators with single precision. Different strategies can be used to choose the best way an approximation can be evaluated, taking into account speed and area constraints and trade-offs. Combining applications like Maple and synthesis tools for FPGAs, an integrated environment is being developed to automatically generate an hardware solution that satisfies user specified parameters like precision and speed/area trade-offs.

Future work can be done in two main directions. The first one, refers to the elementary function approximation itself. The exposed methods can be combined with table based methods for a mixed solution. Because these tables can be stored inside the FPGAs using the look-up tables available

in the CLBs this seems to be an interesting alternative to the methods described. The other direction concerns the way how the arithmetic operators are implemented and how they can be optimized for area and performance constraints.

REFERENCES

[Bak75] G. A Baker. *Essentials of Padé approximants*. Academic Press, New York, 1975.

[Boa85] I. S. Board. *IEEE Standard for Binary Floating Point Arithmetic*. The Institute of Electrical and Electronics Engineers, New York, 1985. ANSI/IEEE Std. 754-1985.

[CGG+91] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Library Reference Manual*. Springer Verlag, Berlin, Germany, 1991.

[HCL+68] J. F. Hart, E. W. Cheney, C. L. Lawson, H. J. Maehly, C. K. Mesztenyi, J. R. Rice, H. G. Thacher, and C. Witzgall. *Computer Approximations*. Wiley, New York, 1968.

[Knu81] Donald Knuth. *The Art of Computer Programming – Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, MA, 1981.

[Kor98] Israel Koren. *Computer Arithmetic Algorithms*. Brookside Court, Amherst, MA, 1998.

[Mul97] Jean-Michel Muller. *Elementary Functions – Algorithms and Implementation*. Birkhäuser, Boston, 1997.

[New96] Bernie New. Using the dedicated carry logic in XC4000E. Application Note XAPP 013, Xilinx, 1996.

[Ric64] J. R. Rice. *The Approximation of Functions*. Addison-Wesley, Reading, MA, 1964.

[Xil96] Xilinx. *The Programmable Logic Data Book*, 1996.