Evaluating Elementary Functions in a Numerical Coprocessor Based on Rational Approximations

ISRAEL KOREN, SENIOR MEMBER, IEEE, AND OFRA ZINATY

Abstract—**High-speed numerical coprocessors for fixed-point and floating-point arithmetic operations are now available due to recent advances in VLSI technology. Some of these coprocessors are also capable of evaluating elementary functions (logarithm, exponential, trigonometric, inverse trigonometric, etc). Most commonly used methods for hardware evaluation of these functions are based on simple iterative equations, involving only shift and add operations. Their major drawback is their linear convergence which slows down the calculation especially for high-precision floating-point operands.**

In this paper, we examine a different approach to hardware evaluation of elementary functions for high-precision floating-point numbers (in particular, the extended double precision format of the IEEE standard P754). The evaluation is based on rational approximations of the elementary functions, a method which is commonly used in scientific software packages.

We present a hardware model of a floating-point numeric coprocessor consisting of a fast adder and a fast multiplier, and add to it minimum hardware required for evaluation of the elementary functions. Next, we derive rational approximations for evaluating the elementary functions and test the accuracy of the results. We then estimate the calculation time of these approximations in the proposed numeric processor. Our final conclusion is that rational approximations can successfully compete with previously used methods when execution time and silicon area are considered.

Index Terms—**Cordic, elementary functions, numeric coprocessor, rational and polynomial approximations.**

I. Introduction

ADVANCES in VLSI technology have enabled the design and implementation of high-speed chips for complex arithmetic operations. There are two types of such arithmetic chips. The first type includes chips capable of executing only a limited number of operations like add, subtract, multiply, and divide. Examples for this type are the Weitek 2265 Adder and the Weitek 2264 Multiplier, the AMD 29325 which contains both adder and multiplier for single precision floating-point numbers, and the Analog Devices ADSP 3212 and 3222 which are single and double precision multiplier and ALU (arithmetic logic unit), respectively.

The second type of arithmetic chips consists of general-purpose numerical coprocessors which execute a rich repertoire of floating-point operations. Examples of this type are Intel 80287, Motorola 68881, National Semiconductor 32081, and alike. Most of these general-purpose numerical coprocessors support the double extended format of the IEEE standard for floating-point numbers [9]. The double extended format consists of 1 sign bit, 15 bit biased exponent, and at least 64 bits of mantissa for a total of (at least) 80 bits.

Many of the chips of the second type are also capable of evaluating elementary functions. We concentrate in this paper on elementary functions and analyze algorithms for their calculation. The most commonly used methods for hardware evaluation of these functions are the "Cordic" type methods. The original Cordic method was developed by Volder [15] and then generalized by Walther [16]. Similar techniques were later on developed, e.g., [13], [3], and several implementations suggested [1], [14]. All are based on simple iterative equations, involving only shift and add operations and were developed in an effort to avoid the time consuming multiply and divide operations. The major disadvantage of the Cordic type algorithms is their linear convergence resulting in an execution time which is linearly proportional to the number of bits in the operands. Several speedup techniques were therefore proposed (e.g., [3], [5]), increasing the amount of hardware needed.

Recent advances in VLSI technology have reduced significantly the time penalty involved in executing multiplication and division compared to add/subtract operations. For example, the above mentioned Weitek chips execute multiplication at the same speed as addition; the latency of a single add or multiply operation for single precision (i.e., 32 bit) floating-point numbers is 280 ns and it is 320 ns for double precision (i.e., 64 bit numbers). These two operations have a pipeline throughput of one result every two cycles (80 ns). The 2264 chip is capable of executing single precision division in 600 ns and double precision division in 800 ns. The Analog Devices chips also execute multiplication and addition at the same speed. The pipeline throughput for both operations is one result per cycle (50 ns) for single or double format operation. The latency is 130 ns for single precision and 155 ns for double precision. The multiplier is also capable of executing single and double precision division in 300 and 600 ns, respectively.

Also, the high density of VLSI chips now enables the use of high-precision floating-point formats such as the 80 bit double

Manuscript received November 7, 1989; revised March 8, 1990.

I. Koren is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003.

O. Zinaty is with the Department of Electrical Engineering, Technion-Israel Institute of Technology, Haifa 32000, Israel.

IEEE Log Number 9036139.

extended IEEE format. For such formats the linear convergence of the Cordic type methods slows down the calculation considerably. Consequently, algorithms for hardware evaluation of elementary functions which require a smaller number of steps even if some of them involve multiplication or division may now be more attractive than the Cordic type algorithms.

One possible approach to hardware evaluation of elementary functions is based on polynomial or rational approximations of these functions, a method which is common in scientific software packages. Hwang *et al.* [8] have presented a systolic pipeline implementation of Chebyshev polynomial approximations. We study in this paper rational approximations and examine the accuracy of the results obtained when these approximations are used for the evaluation of elementary functions.

In the next section, a hardware model for a numerical coprocessor capable of executing this type of algorithm is presented. Polynomial and rational approximations for some elementary functions are derived in Section III and their accuracy is tested. In Section IV, the execution time of these approximations is estimated and then compared to other methods. Final conclusions are presented in Section V.

II. The Hardware Model for the Numerical Coprocessor

Our conceptual model of a general-purpose coprocessor is depicted in Fig. 1. It has a microprogram control unit and contains two major arithmetic units, a floating-point adder and a floating-point multiplier. The floating-point adder unit consists of two adders: a mantissa adder and a small (15 bit) exponent adder. The floating-point multiplier unit contains a mantissa multiplier and an exponent adder. These two units have separate flags like sign, sign of exponent, zero, zero fraction, infinity, and NAN (Not A Number) [9]. In addition, the coprocessor contains a bus interface unit (BIU), an instruction register (IR), and decode logic and an address generator (for the microcode ROM). There is also a register file consisting of eight registers accessible by the user and ten internal registers ($R0$–$R9$), four of which are connected to the adder ($R0$–$R3$), and four to the multiplier ($R4$–$R7$). These ten registers allow concurrent execution of add and multiply operations.

To these functional units we add a relatively small amount of hardware necessary to support the calculation of elementary functions. This includes some extra microinstructions in the control ROM and a separate coefficient ROM with less than 100 locations to hold the coefficients for the rational or polynomial approximations and the constants needed for the argument reduction step.

There is also a separate conversion unit which converts the single and double formats of the incoming floating-point operands to the double extended format which is the only one used internally. It also converts the final results to the proper external format. The use of a larger internal format increases the accuracy of all intermediate calculations which is especially important for the complex elementary functions.

III. Deriving Polynomial and Rational Approximations

All continuous elementary functions can be approximated by either a polynomial of degree m, $P_m(x)$, or a rational

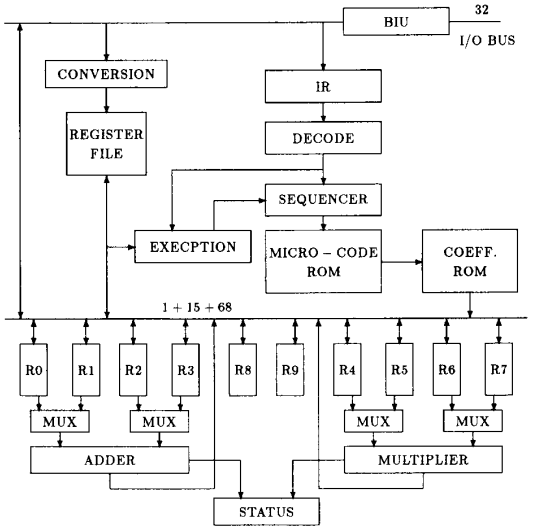


Fig. 1. Block diagram of the numerical coprocessor.

expression $R_{mn}(x)$ where m and n are the degrees of the numerator and denominator polynomials, respectively. A polynomial approximation is a special case of a rational one, $P_m(x) = R_{m0}(x)$, and consequently, only the more general case of rational approximations will be discussed in what follows. The accuracy of the rational approximation is determined by the degrees m and n and by the values of the coefficients. To reduce the number of coefficients (and thus speed up the calculation) and avoid singular points, an argument interval $[a, b]$ which is smaller than the function domain is in many cases used. An algorithm for evaluating elementary functions using rational approximation, therefore includes in principle, the following three steps:

1) reduction of the argument to a predetermined approximation interval ($[a, b]$)

2) evaluating the rational approximation of the reduced argument

3) obtaining the final result.

The coefficients of the rational approximation $R_{mn}(x)$ for a given function $f(x)$ are selected so as to minimize the maximum relative error in the interval $[a, b]$:

$$\min_{R_{mn}(x) \in V_{mn}[a,b]} \max_{[a,b]} \left[\frac{R_{mn}(x) - f(x)}{f(x)} \right]$$

where $V_{mn}[a, b]$ is the set of all rational expressions with numerator and denominator of degrees m and n (or less), respectively. According to Chebyshev's theorem [6] there is a unique rational approximation which minimizes the above relative error. Given a function $f(x)$ and the desired accuracy, we have to choose the approximation interval $[a, b]$, select proper values for m and n (which implies deciding on either a rational or a polynomial approximation), and finally calculate the coefficients of the approximation which minimizes the maximum relative error.

For the elementary functions there is a natural selection of the intervals $[a, b]$ which simplifies the step of argument

reduction. This step is based on properties like periodicity (e.g., $\sin(2\pi k + x) = \sin x$), symmetry (e.g., $\cos(-x) = \cos x$), additivity (e.g., $\ln(x \cdot y) = \ln x + \ln y$) and alike. Consequently, the following intervals are commonly selected [7].

1) For sin, cos, and tan, $[a, b] = \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$.
2) For ln, $[a, b] = \left[\frac{1}{\sqrt{2}}, \sqrt{2}\right]$.
3) For $\sin^{-1}$, $\cos^{-1}$, $[a, b] = \left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$.
4) For $\tan^{-1}$, $[a, b] = [-1, 1]$.
5) For 2^x, $[a, b] = \left[0, \frac{1}{2}\right]$.

We next have to decide between rational and polynomial approximations. Rational approximations are usually preferred, since they are in most cases more accurate than polynomial approximations when the same number of coefficients is used [6], [7]. More importantly, rational approximations are more suitable to our model of a numeric coprocessor since they can take advantage of the high level of parallelism available in the coprocessor. An add operation for evaluating the numerator and a multiply operation for evaluating the denominator (and vice versa) can be performed in parallel. To further increase the level of concurrency and thus reduce the total calculation time, we prefer rational approximations for which the degrees of the numerator and denominator are either the same or differ by at most one. Note that, unlike polynomial approximation, the rational approximation requires a divide operation whose execution time is higher than that of addition or multiplication. In currently available technology, however, the execution time of division is sufficiently low (e.g., 12 cycles in ADSP 3212, 24 cycles in Weitek 2264) and thus, the rational approximation is preferable.

Finally, we have to determine the degrees m and n, and calculate the coefficients of the best rational approximation (i.e., the *minmax* approximation) so that the required accuracy is achieved. The IEEE floating-point standard [9] does not define error bounds for the elementary functions (except square root). We attempt, however, to achieve the same accuracy as required for the basic floating-point operations, i.e., the error in round to nearest mode should be less than half of the least significant bit of the mantissa. (The latter is commonly known as *ulp*—unit in the last place.) For the double extended format this means that the error should be less than 2^{-65}.

For each elementary function we have examined several choices of m and n for the rational approximation. Unfortunately, for given m and n there is no analytical method to calculate the coefficients of the best rational approximation and numerical methods must be used [6]. These methods do not always converge and in many cases only "nearly optimal" rational approximations can be generated. The coefficients for many rational approximations for various accuracies have been generated in the past and are tabulated in [7]. To provide a greater flexibility in selecting rational approximations, we have developed a program for calculating the required coefficients. The program allowed us to generate rational approximations considering simplicity of implementation (based on the hardware model presented in Section II), in addition to accuracy. Our program is based on the Second Algorithm of Remez [6], [12] which yields the best rational approximation. This algorithm, in principle, searches for $m + n + 2$ points $a \le x_1 < x_2 < \cdots < x_{m+n+2} \le b$ for which the relative error is maximal and determines the coefficients for which the relative errors in any two adjacent points x_i and x_{i+1} have the same value but opposite signs. For more details the reader is referred to [6] and [12].

To select the most appropriate rational approximation for each elementary function we then tested the accuracy of the rational expressions generated by our program and those derived by Hart *et al.* [7]. The accuracy of any rational approximation is affected by errors introduced in the various steps of the calculation which in turn depend on the precision of the intermediate hardware operations. Consequently, to determine the accuracy of a given rational approximation we have simulated the operation of the numerical coprocessor (as presented in Section II) for several lengths of the internal representation. This allows us to determine the minimal length of the internal representation which is required to achieve the desired accuracy for a given rational approximation.

Every rational approximation was checked in three lengths of mantissas: 67, 68, and 69 bits. The IEEE standard adds 3 bits to the intermediate mantissa to enable a correct final result, namely, the guard, round, and sticky bits. For the extended double precision format this implies that the length of the intermediate mantissa should be at least 67 bits. We did not go beyond 69 bits because of low cost effectiveness where cost is measured in size of silicon area.

To avoid exhaustive simulation for all possible arguments we restricted the simulation to several specially selected arguments [4] and randomly selected arguments from the approximation interval and from other intervals. The computed results were then compared to the "correct" results in the same format. The specially selected arguments include, for example, the largest and smallest arguments allowed [4].

The correct results were obtained using the Multiple Precision (MP) package developed by Brent [2]. This software package is capable of calculating all basic floating-point operations and all elementary functions in almost every precision. The high-precision results produced by this package can then be rounded to the selected width of the internal coprocessor operation. The algorithms for elementary function evaluation in the MP package are based on power series in special subintervals for exp and ln, and on Taylor series for tan, $\sin^{-1}$, and $\tan^{-1}$.

When testing the approximation error for random arguments we had to determine the size of the random argument test set (in the approximation interval). The number 2000 is recommended in [4], and to check whether it is sufficient for the high-precision operands in our case, we have examined test sets consisting of 1000, 2000, and 3000 random arguments in the approximation interval for the functions sin and ln. The results of this test are given in Table I showing the percentage of wrong results when rounding to 65 bits is performed. The conclusion that can be drawn from this table is that when the number of arguments is increased beyond 2000, the results

TABLE I
ERRORS IN ROUND TO 65 BITS FOR DIFFERENT SIZES OF THE RANDOM TEST SET

Function	Mantissa's length	1000 arguments	2000 arguments	3000 arguments
sin	67	17.10%	18.55%	18.30%
sin	68	7.40%	9.15%	9.07%
sin	69	5.80%	5.95%	6.07%
ln	67	39.80%	43.40%	42.90%
ln	68	20.30%	21.65%	21.50%
ln	69	9.30%	10.80%	9.93%

change by less than one percent. Consequently, 2000 random arguments were selected for every elementary function in its approximation interval.

In what follows, we present in detail the alternatives examined for the function tan. Similar analysis was conducted for all elementary functions. The function tan was evaluated using

$$\tan \frac{1}{4}\pi x \approx x\frac{P_m(x^2)}{Q_n(x^2)} \qquad \text{where } -1 \le x \le 1.$$

The following four sets of coefficients were examined. The first three are from Hart *et al.* [7] while the coefficients for the fourth were generated by our program and appear in the Appendix.

a) *H*34—[7] (index 4268), numerator of degree 3 and denominator of degree 4.

b) *H*43—[7] (index 4258), numerator of degree 4 and denominator of degree 3.

c) *H*44—[7] (index 4287), numerator of degree 4 and denominator of degree 4.

d) *P*44—[17], numerator of degree 4 and denominator of degree 4 (see the Appendix).

$$P44 = x \cdot \frac{P_4(x^2)}{Q_4(x^2)} = x \cdot \frac{(((a_4x^2 + a_3)x^2 + a_2)x^2 + a_1)x^2 + a_0}{(((x^2 + b_3)x^2 + b_2)x^2 + b_1)x^2 + b_0}.$$

Figs. 2 and 3 show the error of these four alternatives using 67 and 68 bit mantissas. The error in these figures is normalized by 2^{-64}. Thus, a normalized error of -0.25 in Fig. 2 means that the calculated error was smaller by $0.25 \cdot 2^{-64}$ than the correct value. In this figure, the mantissa is 67 bits long and consequently, the smallest (nonzero) normalized error is $2^{-67}/2^{-64} = 0.125$. From Fig. 2 we can see that the errors using all four alternatives in 67 bit mantissas exceed $0.5 \cdot 2^{-64} = 2^{-65}$. Since in addition to the error within the approximation interval we should also expect errors in the argument reduction process, we decided to examine the use of 68 bit mantissas.

In Fig. 3, we see that the smallest error in 68 bit mantissa is achieved by the rational expression *P*44. Its worst case is 2^{-66} and it was, therefore, chosen for approximating tan. The 68 bit mantissa satisfies our demand for accuracy and there is no need to further increase the number of mantissa bits.

We would like to emphasize that there is only a slight improvement to the accuracy, achieved by rational expressions with higher degrees of the numerator and the denominator. This is due to the fact that higher degrees require more operations that in turn may increase the error.

Similar tests were conducted for all elementary functions. For each function we have generated, through our program, coefficients for several rational approximations. In all cases, we have attempted to select equal degrees for the numerator and denominator to exploit the parallelism between the multiplier and adder in the numeric coprocessor. Also, we made an effort to use the same degrees of numerator and denominator for as many functions as possible to reduce the number of microcode routines that have to be added for the evaluation of the elementary functions. We then compared the accuracy of the approximations generated by our program to that of the corresponding rational approximations in [7]. Finally, the rational approximations yielding the smallest errors have been selected and their errors (for the functions ln, $\sin^{-1}$, and 2^x in 67 and 68 bit mantissas) are shown in Figs. 4–6. Here too the error is normalized by 2^{-64}. For the functions sin, cos, tan, and ln rational approximations with degree 4 in the numerator and in the denominator were selected. For the inverse trigonometric functions we had to select approximations with degree 7 in the numerator and in the denominator.

The general expressions for the above functions are shown below and the appropriate coefficients appear in the Appendix.

$$\sin \frac{1}{4}\pi x \approx x\frac{P_4(x^2)}{Q_4(x^2)} \qquad \text{where } -1 \le x \le 1$$

$$\cos \frac{1}{4}\pi x \approx \frac{P_4(x^2)}{Q_4(x^2)} \qquad \text{where } -1 \le x \le 1$$

$$\ln x \approx z\frac{P_4(z^2)}{Q_4(z^2)} \qquad \text{where } z = \frac{x-1}{x+1}; \quad \frac{1}{\sqrt{2}} \le x \le \sqrt{2}$$

$$\sin^{-1} x \approx x\frac{P_7(x^2)}{Q_7(x^2)} \qquad \text{where } -\frac{1}{\sqrt{2}} \le x \le \frac{1}{\sqrt{2}}$$

$$\cos^{-1} x = \frac{\pi}{2} - \sin^{-1} x \qquad \text{where } -\frac{1}{\sqrt{2}} \le x \le \frac{1}{\sqrt{2}}$$

$$\tan^{-1} x \approx x\frac{P_7(x^2)}{Q_7(x^2)} \qquad \text{where } -1 \le x \le 1.$$

For the functions sin, cos, and 2^x, even an implementation with 67 bits of mantissa yields the desired accuracy. The error of $\sin^{-1}$ (shown in Fig. 5) is the worst of all the functions. It reaches almost 2^{-63} in 68 bit mantissa.

The function 2^x for which none of the above possibilities is suitable can be implemented using either a rational approximation of degree 5 in the numerator and the denominator, or a different special form [7]. The first alternative is

$$2^x \approx \frac{P_5(x)}{Q_5(x)} \qquad \text{where } 0 \le x \le 0.5.$$

The error of this rational approximation is shown in Fig. 6.

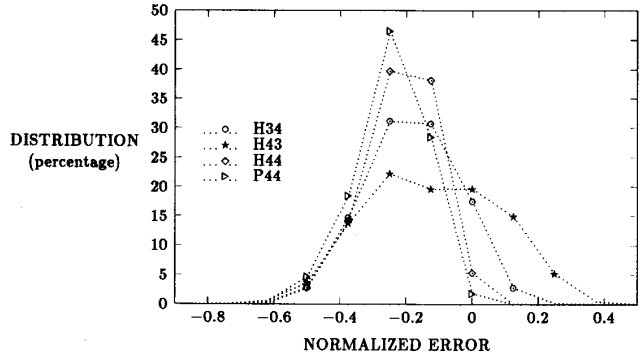


Fig. 2. Error distribution for the four rational approximations for tan in 67 bit mantissa: (a) $H34$, (b) $H43$, (c) $H44$, and (d) $P44$.

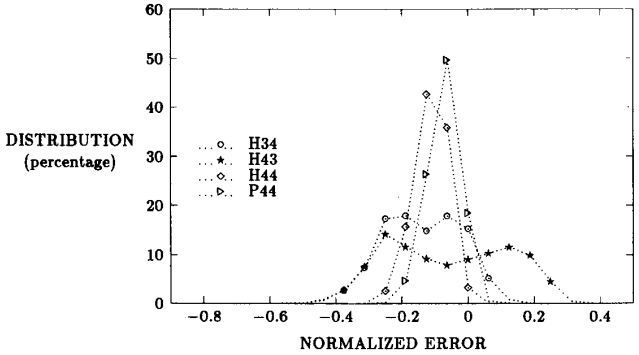


Fig. 3. Error distribution for the four rational approximations for tan in 68 bit mantissa: a) $H34$, b) $H43$, c) $H44$, and d) $P44$.

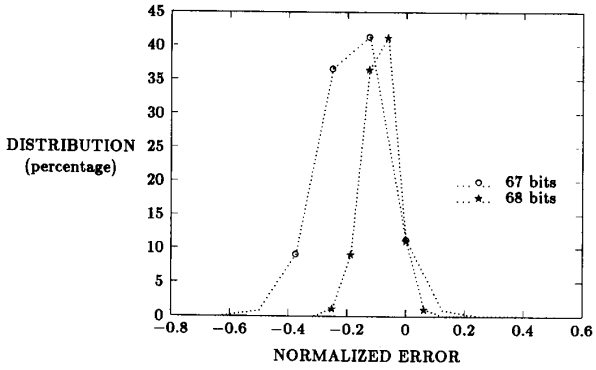


Fig. 4. Error distribution for the ln function in a) 67 bits and b) 68 bits.

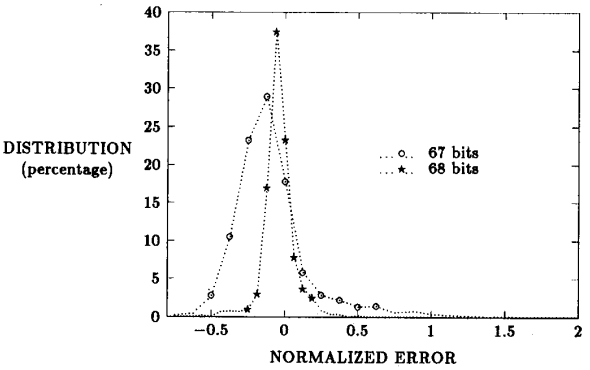


Fig. 5. Error distribution for the $\sin^{-1}$ function in a) 67 bits and b) 68 bits.

A faster implementation was presented in [7]:

$$2^x \approx \frac{Q_3(x^2) + xP_2(x^2)}{Q_3(x^2) - xP_2(x^2)} \qquad \text{where } 0 \le x \le 0.5$$

and its error distribution is very similar to the one shown in Fig. 6. The function 2^x is often implemented since it allows the calculation of other exponential functions. For example, to evaluate e^y we find an integer N and a fraction x so that $y \log_2 e = N + x$ and then $e^y = 2^{N+x} = 2^N \cdot 2^x$. The term 2^N is incorporated into the exponent field of the floating-point number. If the resulting fraction x is larger than 0.5 we use instead $e^y = 2^{N+x} = 2^N \cdot 2^{0.5} \cdot 2^{(x-0.5)}$, requiring the precalculation of the constant $\sqrt{2}$.

Considering the results for all the elementary functions we recommend the length of 68 as the length of the mantissa for intermediate results. This length is essential for the functions $\sin^{-1}$, tan, and ln to achieve the required accuracy even without considering possible errors due to the reduction process. This compares well to the 67 bit data path in the Intel 8087 which uses the Cordic algorithm and guarantees an error not to exceed 2^{-63} [10].

When testing the accuracy of all the above approximations

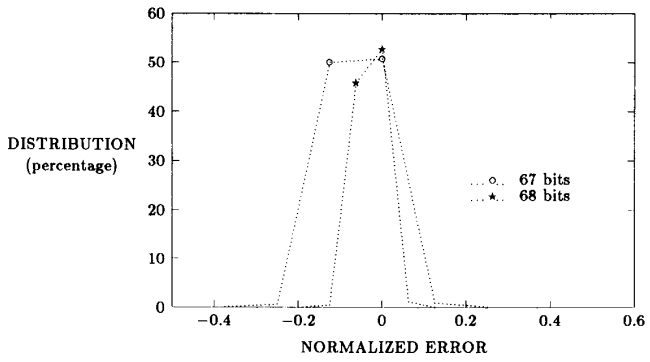


Fig. 6. Error distribution for the function 2^x in a) 67 bits and b) 68 bits.

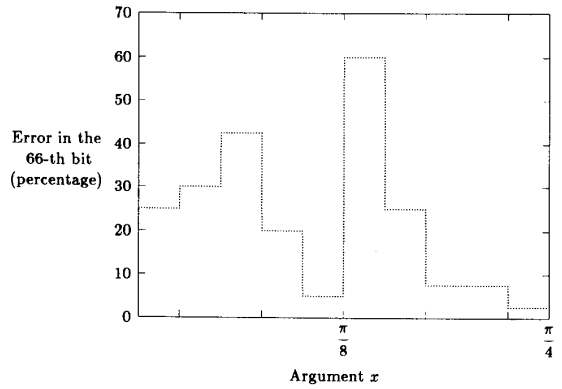


Fig. 7. Errors in subintervals for the sin function.

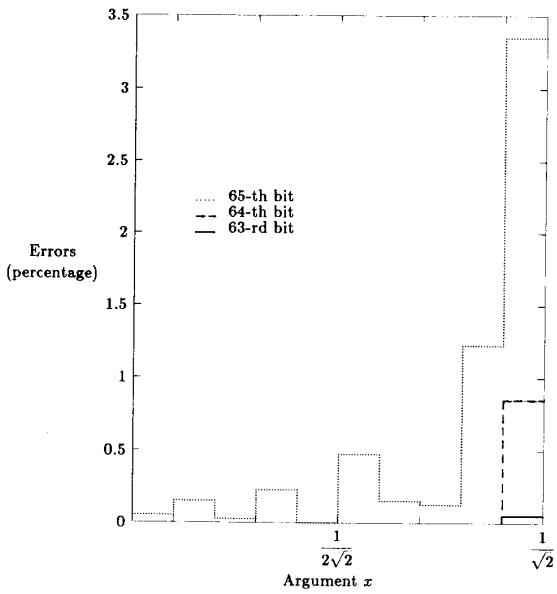


Fig. 8. Errors in subintervals for the $\sin^{-1}$ function, a) In the 65th bit, b) in the 64th bit, and c) in the 63rd bit.

we have also investigated how the error is distributed in the approximation interval. We divided the approximation interval of every function into ten equal subintervals. In every such subinterval, 200 random arguments were evaluated. For most of the functions, approximately uniform distribution was observed. For $\sin^{-1}$ the errors in the last approximation subintervals were more frequent and of larger magnitude than in the other subintervals.

An example of an error distribution graph of the sin function is depicted in Fig. 7. In this graph we can see the percentage of errors in the 66th bit. This is the worst error in a test of 2000 arguments in the approximation interval. We also show the error distribution graph for the function $\sin^{-1}$ in Fig. 8. This graph includes three curves: one for the percentage of errors in the 65th bit, one for the errors in the 64th, and one for the errors in the 63rd bit.

IV. Estimating the Execution Time

To estimate the execution time of the derived rational approximations in the coprocessor presented in Section II, detailed microcode routines were developed for all the elementary functions mentioned above [17]. The execution time (in clock cycles) of the various basic operations were estimated based on the execution times of the same operations in the Weitek 2264 and 2265 arithmetic chips. For example,

$$T_{\text{add}} = 2, \quad T_{\text{mul}} = 2, \quad T_{\text{div}} = 24.$$

In what follows, we estimate the execution time for the rational approximation of the function tan. As we have seen in Section III, a numerator and denominator of degree 4 are needed to achieve the required accuracy with 68 bit mantissa. The algorithm for calculating $\tan(\alpha)$ consists, therefore, of the following steps.

1) Find an integer N and a fraction x such that $\alpha \cdot \frac{4}{\pi} = 2N + x$. This is useful since

$$\tan(\alpha) = \tan\left(\frac{1}{4}\pi\frac{4}{\pi}\alpha\right) = \tan\left(\frac{1}{4}\pi(2N+x)\right)$$

and consequently

$$\tan(\alpha) = \begin{cases} \tan\left(\frac{1}{4}\pi x\right) & \text{if } N \text{ is even} \\ \dfrac{-1}{\tan\left(\frac{1}{4}\pi x\right)} & \text{if } N \text{ is odd.} \end{cases}$$

2) Calculate x^2.

3) Calculate $P_4(x^2)$ and $Q_4(x^2)$ with the appropriate coefficients for tan (see the Appendix).

4) If N is even then $\tan(\alpha) = \dfrac{x \cdot P_4(x^2)}{Q_4(x^2)}$

If N is odd then $\tan(\alpha) = \dfrac{-Q_4(x^2)}{x \cdot P_4(x^2)}$.

In step 3), which is repeated in the procedure for other elementary functions, we take advantage of the separate adder and multiplier in the coprocessor which can operate in parallel. The two polynomials $P_4(x^2)$ and $Q_4(x^2)$ are evaluated concurrently using Horner's method [6]. Assuming that $T_{\text{mul}} \geq T_{\text{add}}$ then the additions are executed in parallel to the multiplications and the time (in clock cycles) needed to calculate the two polynomials is

$$T_{PQ} = 8T_{\text{mul}} + T_{\text{add}} + 11T_{\text{mov}}$$

where T_{mov} is the time of a move operation from one register to the other and is estimated to be one cycle. According to the microcode routine for tan, the execution time for arguments in the approximation interval ($x \in [-\pi/4, \pi/4]$) is

$$26T_{\text{mov}} + 7T_{\text{add}} + 10T_{\text{mul}} + T_{\text{div}} + 2T_{\text{rem}} + 3T_{\text{test}}$$

where T_{rem} is the time to calculate the remainder x when α is in the approximation interval and T_{test} is the time needed to test whether a number is zero or positive. These two are estimated to be 10 and 1 cycle, respectively.

Based on the estimates for the basic operations we estimate the evaluation time of tan to be 107 cycles. For arguments in $[-2\pi, 2\pi]$ one should add 1 cycle because the remainder operation is longer. Similarly, every doubling of the interval $[-2\pi, 2\pi]$ adds 1 more cycle.

The execution time of the FTAN instruction in the MC68881 for an argument in $[-\pi, \pi]$ is 442 cycles and it is 450 cycles in the Intel 8087. This is the time from the end of the input operand conversion (to the internal extended double precision format) until the calculation is complete. It does not include the time needed for rounding and exception handling. We should not, however, compare the 442 (or 450) cycles to the 107 cycles needed to calculate tan using a rational approximation, since the Motorola and Intel chips incur extra overhead due to their extended functionality. Instead, we may compare the 107 cycles to the number of cycles required by a direct implementation of a Cordic type algorithm. A single Cordic iteration for tan takes at least 3–4 cycles for a total of 201–268 cycles for 67 iterations. For example, a pipelined implementation of a Cordic algorithm reported by Naseem and Fisher [11] requires $3k + 2$ cycles where k is the number of mantissa bits. This number of cycles is considerably higher than our estimate of 107 cycles. We may conclude, therefore, that if the implementation of a numerical coprocessor with separate ALU and multiplier/divider array within a single chip is feasible, then the use of a rational approximation for the tan function can speed up substantially its evaluation. Note, however, that this conclusion holds only for high-precision floating-point formats. For single-precision floating-point numbers (i.e., 32 bits), the time to evaluate the rational approximation is still close to 100 cycles while the execution time of a Cordic type algorithm reduces to between 72 and 96 cycles.

Similar calculations were done for the other elementary functions but the details of these are not presented here for the sake of brevity. For all of them, the estimated execution tiime was very close to that of the tan function, i.e., around 100 cycles.

V. Conclusions

The possibility of evaluating elementary functions using rational approximations for high-precision floating-point operands was examined in this paper. Our results show that high accuracy can be achieved with properly selected rational approximations. Moreover, we can expect faster evaluation of elementary functions for high-precision operands if we take advantage of the currently available technology and design a numerical coprocessor with separate ALU and multiplier/divider array.

Acknowledgment

Discussions with A. Sidi are gratefully acknowledged. The authors would like also to thank the anonymous reviewers for their helpful comments.

Appendix

The Coefficients for the Rational Approximations

This Appendix contains the coefficients that were produced by our program for the analyzed elementary functions. The coefficients for arctan were taken from [7] and are not reproduced here.

SINE

The accuracy: 81.35 binary digits.

a_0 = 1805490264.690988571178600370234394843221

a_1 = -164384678.22749983772612961258795266051 1

a_2 = 3664210.64758126181022792446516082736 5

a_3 = -28904.14024646178135722374193598009 7

a_4 = 76.56898108871740581013254352368 2

b_0 = 2298821602.63892266208648752033082725117 2

b_1 = 27037050.11889443677662486664823559198 8

b_2 = 155791.38854694769320646942397950567 1

b_3 = 540.56750126128402476777928070008 9

b_4 = 1.00000000000000000000000000000

COSINE

The accuracy: 75.84 binary digits.

a_0 = 1090157078.17487142042884901726254903860 6

a_1 = -321324810.99315071240135295939764854168 1

a_2 = 12787876.84952387894405188532559387817 7

a_3 = -150026.20604594811056831088716640597 2

a_4 = 538.33356420318266166431915137945 1

b_0 = 1090157078.17487142042886729567003950688 6

b_1 = 14907035.77664387976741096950962840650 2

b_2 = 101855.81194366136830260814669508221 8

b_3 = 429.77286510739182324567126448931 1

b_4 = 1.00000000000000000000000000000

TANGENT

The accuracy: 77.93 binary digits.

a_0 = 4131609.17077946361283161369760968866 4
a_1 = -349892.44618982737945619417450261116 0
a_2 = 6171.94188939819385408877010598385 7
a_3 = -27.95279487296424998280322448100 0
a_4 = 0.01751083054355804551890675686 7
b_0 = 5260528.17962686727108291354464503247 5
b_1 = -1527149.65042842324751200579788073019 7
b_2 = 54978.80220191476978882579202597858 1
b_3 = -497.60020536676868221418995309566 55
b_4 = 1.00000000000000000000000000000 0

LN

The accuracy: 80.26 binary digits.

a_0 = 75.15185614991079464273237545292 8
a_1 = -134.73039968865933984458672116291 4
a_2 = 74.20110142063425732649900827551 5
a_3 = -12.77714340149074010375840645432 3
a_4 = 0.33257960182438920615106352997 1
b_0 = 37.57592807495539732136615600778 1
b_1 = -79.89050920264813569590999552131 0
b_2 = 56.21553482954209427714341740471 1
b_3 = -14.51697119505668294871912566171 7
b_4 = 1.00000000000000000000000000000 0

2^x

The accuracy: 71.72 binary digits.

a_0 = -206059.51365146241730060320610576260 8
a_1 = -72102.25779558823052518632485717604 5
a_2 = -11240.02876510674774928628566581423 4
a_3 = -989.02784689063694455173596338784 5
a_4 = -49.98982724072861359957320341432 1
a_5 = -1.18920711500272106594716056758 0
b_0 = -206059.51365146241730068784239604636 1
b_1 = 70727.31311947650507364076097089355 9
b_2 = -10763.50925227037618524880103429942 1
b_3 = 918.24250408819861089603436213143 3
b_4 = -44.53626652588117935627275274414 3
b_5 = 1.00000000000000000000000000000 0

ARCSINE

The accuracy: 68.3 binary digits.

a_0 = -972.78220770922834172920799159383 9
a_1 = 3498.39665059260002154231018423922 9
a_2 = -4995.83859894348078623005303885314 7
a_3 = 3590.01790500438623258807553276092 4
a_4 = -1352.47420407153663600084332681300 8
a_5 = 250.60515802144403620851358695308 8
a_6 = -18.43991246936710793725330665902 6
a_7 = 0.26207654320832171506209050286 1
b_0 = -972.78220770922834172492795479452 3
b_1 = 3660.52701854413807567824108222022 7
b_2 = -5532.96776978931135997044718915442 7
b_3 = 4281.06745070832351020219144000569 8
b_4 = -1784.87423288700660129199074251193 7
b_5 = 384.56093754799195602247327382722 3
b_6 = -36.69803011109749911881831447857 2
b_7 = 1.00000000000000000000000000000 0

References

[1] P. W. Baker, "Suggestion for a fast binary sine/cosine generator," *IEEE Trans. Comput.*, vol. C-25, pp. 1134–1136, Nov. 1976.

[2] R. P. Brent, "MP User's guide," Fourth Edition, Tech. Rep. TR-CS-81-08, Dep. Comput. Sci., Australian National Univ., July 1981.

[3] T. C. Chen, "Automatic computation of exponentials, logarithms, ratios and square roots," *IBM J. Res. Develop.*, pp. 380–388, July 1972.

[4] W. J. Cody and W. Waite, *Software Manual for The Elementary Functions*. Englewood Cliffs, NJ: Prentice-Hall, 1980.

[5] M. D. Ercegovac, "Radix-16 evaluation of certain elementary functions," *IEEE Trans. Comput.*, vol. C-22, pp. 561–566, June 1973.

[6] C. T. Fike, *Computer Evaluation of Mathematical Functions*. Englewood Cliffs, NJ: Prentice-Hall, 1968.

[7] J. F. Hart *et al.*, *Computer Approximations*. New York: Wiley, 1968.

[8] K. Hwang *et al.*, "Evaluating elementary functions with Chebyshev polynomials on pipeline nets," in *Proc. 8th Symp. Comput. Arithmetic*, May 1987, pp. 121–128.

[9] "IEEE Standard for Binary Floating Point Arithmetic," ANSI/IEEE 754-1985, also in *IEEE Comput. Mag.*, vol. 14, pp. 51–62, Mar. 1981.

[10] R. Nave, "Implementation of transcendental functions on a numerics processor," *Microprocessing and Microprogramming*, vol. 11, pp. 221–225, 1983.

[11] A. Naseem and D. Fisher, "The modified Cordic algorithm," in *Proc. Symp. Comput. Arithmetic*, May 1985, pp. 144–151.

[12] H. Ralston, *Mathematical Methods for Digital Computers, Vol. 2*. New York: Wiley, 1968, pp. 264–284.

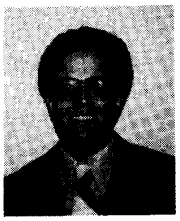
[13] W. H. Specker, "A class of algorithms for $\ln x$, $\exp x$, $\sin x$, $\cos x$, $\tan^{-1} x$ and $\cot^{-1} x$," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 85–86, 1965.

[14] D. G. Steer and S. R. Penstone, "Digital hardware for sine-cosine function," *IEEE Trans. Comput.*, vol. C-26, pp. 1283–1286, Dec. 1977.

[15] J. E. Volder, "The CORDIC trigonometric computing technique," *IEEE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sept. 1959.

[16] J. S. Walther, "A unified algorithm for elementary functions," in *Proc. AFIPS Spring Joint Comput. Conf.*, 1971, pp. 379–385.

[17] O. Zinaty, "Comparison of techniques for evaluating elementary functions," M.Sc. Thesis (in Hebrew), Technion, June 1987.



Israel Koren (S'72-M'76-SM'87) received the B.Sc. (cum laude), M.Sc., and D.Sc. degrees from the Technion–Israel Institute of Technology, Haifa, in 1967, 1970, and 1975, respectively, all in electrical engineering.

He is currently a Professor of Electrical and Computer Engineering at the University of Massachusetts, Amherst. Previously he was with the Departments of Electrical Engineering and Computer Science at the Technion–Israel Institute of Technology. He also held visiting positions with the University of California, Berkeley, University of Southern California, Los Angeles, and University of California, Santa Barbara. He has been a consultant to Digital Equipment Corporation, National Semiconductor, Tolerant Systems, and ELTA—Electronics Industries. His current research interests are fault-tolerant VLSI and WSI architectures, models for yield and performance, floor-planning of VLSI chips, and computer arithmetic. He has edited and co-authored the book, *Defect and Fault-Tolerance in VLSI Systems, Vol. 1* (New York: Plenum, 1989).

Dr. Koren was a Guest Editor for IEEE TRANSACTIONS ON COMPUTERS, special issue on High-Yield VLSI Systems, April 1989.

Ofra Zinaty received the B.Sc. degree in computer engineering and the M.Sc. degree in electrical engineering from the Technion–Israel Institute of Technology in 1984 and 1986, respectively.

Since 1987 she has been a Research Engineer with the Research Institute of the Technion at the Department of Electrical Engineering. Her research interests include color and black and white image processing, VLSI architectures for CCD, and cameras.