Efficient Elementary Function Generation with Multipliers

Hassan M. Ahmed

Electrical, Computer and Systems Engineering
Boston University
Boston, MA 02215

Abstract

Virtually all numerical techniques for elementary function generation share the common property of avoiding multiplication by iteratively performing shift operations. However, with the advent of VLSI, multiplier economics are considerably less formidible than before. We propose combining multipliers with these multiplication free algorithms to construct fast methods of elementary function generation. We demonstrate our idea by combining multipliers with the CORDIC algorithm to achieve fast vector rotation.

I. Introduction

Methods for generating elementary functions were developed several decades ago when special computational elements such as multipliers as well as table storage were both prohibitively expensive. In order to avoid multiplication, these techniques paid the speed penalty associated with iterative shift operations. For example, the CORDIC algorithm [1] is designed for two dimensional vector rotation. Four multiplications can yield a rotation when the required trigonometric quantities are stored to adequate precision. In order to avoid the costs of both storage and multiplication, CORDIC performs a rotation as a sequence of incremental rotations through predetermined angles. The latter are chosen to have tangents that are powers of the machine radix so that multiplications are replaced by shifts and only minimal storage is required. Execution speed is the price paid for these simplifications since a single rotation is replaced by a sequence of incremental rotations. Furthermore, the fina' result must be renormalized, a spurious side effect of CORDIC [1]. Similar tradeoffs appear in other methods of elementary function generation [2]-[4].

Recently, there has been considerable interest in rapid function generation within the signal processing community, since elementary functions appear to be fundamental to signal processing algorithms [5]. Furthermore, DSP processors are often used in graphics applications which require division and square root operations. Unfortunately, the iterative nature of existing function generation methods render them unattractive in the throughput sensitive signal processing and graphics environments. Consequently, parallel and pipelined versions of these algorithms have been actively researched. For example, Naseem et al. [6] have developed a parallel CORDIC by noting that the CORDIC iterations can be decoupled if the incremental rotations are known in advance. Therefore, they concentrate on developing a representation of the angle of rotation in terms of the incremental CORDIC angles. Unfortunately, their decoupling does not apply to the *vectoring* mode, so that many of the CORDIC functions are unattainable through their methods. Ahmed and Fu [7] circumvent this problem with an array CORDIC architecture that parallels array multiplier and divider structures. While they show that their structure offers improved throughput per unit area over multiplier approaches, a VLSI implementation of their method consumes considerable real estate (roughly five times the size of an array multiplier).

In this paper, we offer an alternative approach to elementary function generation that is real estate efficient and exhibits a speed in between the original iterative algorithms and truly parallel structures like [7]. We basically argue that both multiplication hardware and storage are relatively inexpensive (upto a point) given current VLSI technology and therefore, the original efforts to avoid them are no longer justified. By combining these elements with the multiplier-free function generation algorithms, cost effective, high throughput function generation can be achieved. Although this basic principle is general, we restrict our attention to the CORDIC algorithm.

1 This work was supported in part by the National Science Foundation under contract MIP-8705734

II. Just Enough about CORDIC

We assume reader familarity with CORDIC, however we will briefly describe the algorithm for completeness and review some of its convergence properties that are pertinent to this paper.

CORDIC is a numerical method for the rotation of two dimensional vectors that uses a shift and add kernel as its fundamental operation. Let $X_k = [x_k \; y_k]^T$ be the vector at the k^{th} iteration of the algorithm. Then each iteration of CORDIC is given by:

$$X_{k+1} = \begin{bmatrix} 1 & \mu_k m\delta_k \\ -\mu_k \delta_k & 1 \end{bmatrix} X_k$$

$$z_{k+1} = z_k - \mu_k \alpha_k$$

where:

$$\alpha_k = m^{-1/2} \tan^{-1}(\delta_k \sqrt{m})$$

Therefore, the CORDIC algorithm is defined by three scalar equations which we shall refer to as the x_k, y_k and z_k iterations. These equations amount to a rotation of X_k through an angle $\mu_k \alpha_k$ in the coordinate system defined by 'm'. There is also a spurious magnitude change in the vector by a factor:

$$K_k = \sqrt{1 + m\delta_k^2}$$

Clearly, z_k accumulates the total rotation. The sequence $\{\alpha_k\}$ and hence, $\{\delta_k\}$ is fixed and determined a priori in such a way that each element of $\{\delta_k\}$ is an integer power of the machine radix. Therefore, each rotation is simply computed with a shift-and-add kernel.

The convergence properties of CORDIC are essential to the ideas that we present here. Let Φ_i denote the angular error from the desired rotation remaining at iteration i. Then, some key results due to Walther [1] are summarized:

Lemma 1:

Within the domain of convergence:

$$|\Phi_i| < \alpha_{n-1} + \sum_{j=i}^{n-1} \alpha_j \qquad \forall \; i \geq 0$$

Since the sequence $\{\alpha_k\}$ is a declining sequence [1], the angular error becomes progressively smaller. By substituting $i=0$, we obtain the domain of convergence and by substituting $i=n$, we obtain the final angular resolution. These are stated in the following two corollaries also due to Walther [1].

Corollary 1: (Domain of Convergence)

$$|\Phi_0| < \alpha_{n-1} + \sum_{j=0}^{n-1} \alpha_j$$

Corollary 2: (Angular Resolution)

An angular granularity of α_{n-1} is achieved after n iterations.

Further to corollary 2, it is worth noting that the maximum number or iterations is limited by the choice of $\{\delta_k\}$ and the wordlength. For some of the most popular choices of $\{\delta_k\}$ the maximum number of iterations is approximately equal to the wordlength.

Figure 1 shows the CORDIC function set. The interested reader is referred to [1] for further details of CORDIC and the rich set of functions that it generates.

III. Multiplier based CORDIC Computation

CORDIC may be viewed as linearly convergent, in that it generates one bit equivalent of precision per iteration. Hence, the computing speed is strongly dependent on the operand wordlength and the exact precision depends on the choice of the incremental rotation angles, $\{\alpha_i\}$. Consider momentarily, the circular rotation case ($m=1$). The most popular set $\{\alpha_i\}$ achieves an angular resolution of 2^{-n+1} radians in n iterations [1]. Alternatively, a multiplier/accumulator coupled with tables of sines and cosines can compute the rotation in four operations. However, the table must store the trigonometric quantities to 2^{-n+2} radians resolution. At even modest wordlengths, the table can get very large. Apart from the cost of the memory (e.g. its VLSI area), the speed advantage of the multiplier method is eroded by the slower access time of the large memory. Assuming that one quadrant of sines only is stored to the requisite precision, $2^{n-3}\pi$ 'n' bit words are required. By way of example, 32 bit operands would require a 54 Gbit memory! Therefore, while

multipliers and memory are relatively inexpensive given current VLSI technology, such large amounts of memory remain impractical from both a speed and cost viewpoint.

Cost effective function generation can be achieved by combining the CORDIC algorithm with a multiplier. We will present two basic approaches. The first approach is flexible in that it allows one to tradeoff computation speed and storage. The second approach does not offer this flexibility, however it sports the advantage of not requiring any additional memory beyond the iterative CORDIC. We refer to the former approach as an interpolation scheme and the latter as a Taylor Series based approach. Both methods will be named *hybrid CORDIC*.

III.1 CORDIC Functions by Interpolation

Hybrid CORDIC by interpolation is best illustrated by example. Consider the circular rotation CORDIC algorithm ($m=1$). The incremental rotation angles, $\{\alpha_i\}_{i=0}^{n-1}$, are chosen such that $\tan \alpha_i = \delta_i = 2^{-F_i}$. Furthermore, for $m=1$, $\{F_i\}_{i=0}^{n-1} = \{i\}_{i=0}^{n-1}$ results in an angular resolution of 2^{-n+1} radians in n iterations. In order to avoid large memory requirements, let the unit circle be quantized into 2^{-k} parts, $k<n$, yielding an angular resolution of $R=2\pi 2^{-k}/2 \approx 2^{-k+1.65}$. Since this falls short of the 2^{-n+1} radian resolution required of n-bit operands, we will perform the rotation in two steps:

1. Employ table lookup and a multiplier for rotating the original vector with resolution R.

2. Utilize the CORDIC algorithm to refine the resolution to 2^{-n+1} by starting the CORDIC iterations at $i=i_0$ and proceeding to $i=n-1$.

This approach uses the multiplier to perform a coarse rotation and then uses the CORDIC iterations to refine the result. Only $n-i_0$ rather than 'n' CORDIC iterations are neccessary, thereby improving speed over a pure CORDIC approach. Furthermore, the coarser quantization of the unit circle leads to significantly lower storage requirements than the pure multiplier approach (and that in itself improves speed due to faster memory access).

The choice of i_0 is the only outstanding issue and it can be resolved with the aid of Lemma 1 and its corollaries. Note that by ending the CORDIC interpolation with $i=n-1$, we achieve the same angular reolution as if we had simply used the basic CORDIC algorithm. The choice of i_0 hinges on ensuring that the interpolation step leads to convergence. Since step 1 leads to an angular resolution of R, we must select i_0 such that the domain of convergence of step 2 exceeds R. With the aid of lemma 1, this implies:

$$\alpha_{n-1} + \sum_{i=i_0}^{n-1} \alpha_i \geq R \qquad (1)$$

For sufficiently large i, $\alpha_i \approx \delta_i = 2^{-i}$, so (1) becomes:

$$2^{-n+1} + \sum_{i=i_0}^{n-1} 2^{-i} = 2^{-i_0+1} \geq R$$

which implies:

$$i_0 \leq 1 - \log_2 R \qquad (2)$$

For the case above where $R \approx 2^{-k+1.65}$, the CORDIC iteration should be started at $i_0 = k$ or $i_0 = k-1$ and $n-k$ iterations are neccessary. There is clearly a tradeoff between the size of the table of sines and cosines and the speed of the hybrid CORDIC, since increasing the resolution of the table reduces $n-k$. We remark that this hybrid CORDIC technique has a full region of convergence (unlike the original CORDIC) because of the prerotation of step 1.

In order to appreciate the benefits of our hybrid approach, begin with the following definitions:

τ_m is the time required for a multiplication. τ_m includes the memory access time required to retrieve trigonometric quantities and therefore depends weakly on 'k'.

τ_c is the time required to complete a single CORDIC iteration. Typically, $\tau_c < \tau_m$ since the CORDIC iteration requires only a shift and add operation. Occasionally, it will be convenient to compute the shift using the multiplier rather than build separate CORDIC hardware. In this case $\tau_c \approx \tau_m$.

T_m is the rotation computation time using the multiplier and stored table approach

T_c is the rotation computation time using the iterative CORDIC approach

T_h is the rotation computation time using the hybrid multiplier/CORDIC method outlined above

S_c, S_m and S_h are the storage requirements in bits for the three approaches

In what follows, we develop the relative execution speed and storage for the three approaches as a function of n and k. Unfortunately, this comparison depends on the assumptions that are made and we are unaware of any set of assumptions that are universely acceptable. The difficulty with comparing the three approaches lies in determining how many multiplications are neccessary to compute a rotation in the pure multiplier case. Of course we know that this is four (counting a multiply-accumulate as a multiplication), however, CORDIC doesn't strictly speaking, compute a pure rotation, due to the spurious scaling that occurs. The end result of a CORDIC rotation ($m = 1$) through θ is really the transformation:

$$X_{k+1} \approx K \begin{bmatrix} 1 & \tan\theta \\ -\tan\theta & 1 \end{bmatrix} X_k \qquad (3)$$

This obviously requires four multiplications as well but only tangent tables rather than tables of both sines and cosines need be stored. We will assume that the multiplier is used to compute a pure rotation requiring tables of both sines and cosines. Hybrid CORDIC can also achieve a pure rotation with two multiplications for the initial rotation (i.e. equation (3) without the K), followed by the CORDIC iterations, followed by two more multiplications to remove the spurious scale factor[2], i.e. *four* multiplications are required. We will compute our comparisons assuming four multiplications for hybrid CORDIC, realizing full well that the final two multipications are not always required. For example, if one wishes to compute $\cos\theta$ and $\sin\theta$, then the initial vector is set to $X_o = [K^{-1}\ K^{-1}]^T$. Since K is known in advance, $1/K$ requires no explicit calculation. Hence, in this case, hybrid CORDIC does not require the final two multiplications and is faster than our comparisons will indicate.

Under our assumptions, it is straightforward to show that[3]:

$$T_c = n\tau_c \quad and \quad S_c = n^2 \ bits \quad (\ pure\ CORDIC\ )$$

$$T_m = 4\tau_m \quad and \quad S_m = 2^{n-2}\, n\pi \ bits \quad (\ pure\ multiplier\ )$$

$$T_h = 4\tau_m + (n-k)\tau_c \quad and \quad S_h = 2^{k-2}\, n \ bits \quad (\ hybrid\ )$$

We have assumed that only one quadrant of sine and cosine values are stored to the required precision (some preprocessing can reduce the table size by a factor of two at the expense of computation speed). Furthermore, as before, the unit circle is quantized into 2^k parts in the hybrid approach.

The relative merits of the hybrid approach can now be assessed:

$$\begin{cases} \dfrac{T_h}{T_m} = 1 + \dfrac{n-k}{4}\dfrac{\tau_c}{\tau_m} \\[2ex] \dfrac{S_m}{S_h} = 2^{n-k}\,\pi \qquad k \neq 0 \\[2ex] \dfrac{T_h}{T_c} = \dfrac{n - k + 4\dfrac{\tau_m}{\tau_c}}{n} \qquad k \neq 0 \end{cases}$$

These relationships are depicted in Figure 2. The key observation is that while the execution time ratios depend only *linearly* on n-k, the storage requirements vary *exponentially*. Therefore, the hybrid approach offers the potential for drastically reducing the storage requirements of the pure multiplier method while incurring only a moderate speed penalty. The tradeoff of storage and speed can be controlled by the designer's choice of n-k! Furthermore, the hybrid method is faster than the iterative CORDIC. As we noted earlier, reader's disagreeing with our assumptions in forming this comparison may redo the calculations with their own assumptions. Although they will derive different ratios from ours, the key behaviour of exponentially reducing storage requirements while only linearly reducing speed *will still be observed*! This is a fundamental property of hybrid CORDIC.

An example serves to help appreciate the benefits of the hybrid approach. Consider the case of 24 and 32 bit operands and assume k=$n/2$, i.e. approximately half the bits will be generated by the multiplier and half by the CORDIC. Assume further that τ_m/τ_c=2. Table 1 shows the relative storage and speeds of the three approaches. Notice that the large storage reduction occurs at a comparatively modest speed penalty.

---

2 There are other ways of removing the scale factor, see [9] and [10] for example, however these are inferior to simple multiplication when a multiplier is available.

3 Note that the each memory is $n\pi/4R$ bits.

It should be noted that this hybrid CORDIC method is applicable to both custom VLSI architectures as well as to programmed implementation on microprocessors such as DSP processors. In the latter case, it will usually beneficial to use the processor's hardware multiplier as the shifter so $\tau_m/\tau_c=1$.

Note that although we have illustrated our hybrid CORDIC for the circular coordinate system only, extension to the hyperbolic system is straightforward. Finally, we remark that hybrid elementary function generation by interpolation is a general idea and not restricted to the CORDIC illustration that we have provided. Similar techniques can be applied to other elementary function generation methods, for example the convergence computation method of Chen [2].

III.2 A Memoryless Mixed CORDIC Architecture

The previous section presented a mixed approach to CORDIC that allowed us to tradeoff execution speed and storage. In this section, we present another hybrid CORDIC that completely avoids storage of trigonometric tables. Once again, we will illustrate the technique for the circular coordinate system and remark that the extension to the hyperbolic case is straightforward.

While hybrid CORDIC by interpolation first performed a coarse rotation and then used CORDIC iterations to refine the result, in this section, we will do exactly the opposite. CORDIC iterations will be used to first perform a coarse rotation leaving a residual angle, θ_k, following which, a rotation through θ_k will be accomplished using the multiplier. The residual angle is chosen such that a first order Taylor Series approximation of $sin\ \theta_k$ and $cos\ \theta_k$ may be employed. Therefore, rotation by the residual will be accomplished in two multiplications using the multiplier[4].

Proceeding formally, we wish to rotate the vector X_o through an angle z. We begin by executing k CORDIC iterations, where $k<n$. Recall that the remaining angle of rotation is contained in the auxilliary variable, z_k of the CORDIC algorithm [1]. Therefore, $z_k=\theta_k$ and the partially rotated vector will be denoted X_k. Assume that with the aid of lemma 1, k is chosen sufficiently large to admit a first order Taylor Series approximation of $sin\ \theta_k$ and $cos\ \theta_k$, i.e.:

$$\begin{cases} sin\,\theta_k \approx \theta_k \\ cos\,\theta_k \approx 1 \end{cases}$$

Then the final required rotation through θ_k can be approximated as:

$$X_n = \begin{bmatrix} 1 & -\theta_k \\ \theta_k & 1 \end{bmatrix} X_k$$

This final step can be thought of as either a pure rotation through θ_k or a CORDIC transformation as in (3). The two are completely equivalent when θ_k is small since $sin\ \theta_k \approx tan\ \theta_k$ and the scale factor K_k approaches unity.

Once again, a two step hybrid approach has been developed that employs the CORDIC iterations first and terminates with two multiplication steps. No storage tables are needed to complete the final step. The performance of this scheme can be readily seen to be:

$$\begin{cases} \dfrac{T_h}{T_m} = \dfrac{k}{4}\dfrac{\tau_c}{\tau_m} + 1 \\ \dfrac{T_h}{T_c} = \dfrac{k + 4\dfrac{\tau_m}{\tau_c}}{n} \end{cases}$$

(Note, we have assumed that the hybrid method requires four multiplications, two of which are needed to normalize the scale factor of the first k CORDIC iterations).

The final remaining issue is what is k? As we have noted, it should be sufficiently large to admit the series approximation. Assume that all values are stored to n bit precision with the least significant bit having weight 2^{-n+1}.

Proposition 1:

$$k > \left[\frac{n}{2}\right] + 1$$

where $[x]$ denotes the smallest integer exceeding x.

Proof:

Please refer to the appendix.

Proposition 1 indicates that approximately one half of

[4] It is interesting to note that the idea of transforming a function argument, in this case θ_k, into a range where series approximations may be used, is prevalent in other elementary function generation algorithms as well, most notably in the sequential table lookup (STL) technique [3]-[4]. However, employing other elementary function generation methods toachieve this transformation, as we have done here with CORDIC has not been previously attempted.

the resolution of the result is obtained from the final step, while the remainder is due to the CORDIC iterations. Table 2 shows the performance of this method for 24 and 32 bit operands and k=13 and 17 respectively. Comparing with the results of the previous section, we see that this approach offers similar throughput and remarkably better cost because the sine and cosine tables are completely avoided. The VLSI implementation of this latter method is therefore much more compact. We note however, that the method of the previous section offered a wider range of tradeoffs and could be made faster by increasing the storage. No such tradeoff is possible in the Taylor series method.

We have demonstrated the Taylor series based hybrid CORDIC for circular rotation only. The same ideas apply for hyperbolic rotation as well. However, slightly more CORDIC iterations are now required because of the repetition of some of the incremental rotation angles (see [1] for details). For operands less than 120 bits in length, at most three additional CORDIC cycles are neccessary.

We note once again that the Taylor series approach offers remarkably better cost than table based methods because the sine and cosine tables are completely avoided. However, the table methods of the previous section offer a wide range of speed/cost tradeoffs that are not possible in the Taylor series method.

IV. Conclusions

By noting that VLSI technology has reduced the cost of both memory and computational elements, we have developed a hybrid approach to elementary function generation that combines memory, a multiplier and the CORDIC algorithm to perform vector rotation. Both approaches presented offer improved speed over the traditional iterative CORDIC. The stored table method allows designer flexibility in trading storage for speed. The Taylor series method does not offer such flexibility but completely avoids the cost of storage. For equivalent values of k and n, the two methods offer similar throughputs.

Hybrid schemes can also be constructed for methods of elementary function generation other than CORDIC. For example, hybrid convergence computation using Chen's method [2] is also possible. Our hybrid schemes are preferable to methods that employ only series approximations, e.g. [8]. Although slower than completely parallel approaches [7], the hybrid schemes are more compact.

Bibliography

[1] J.S. Walther, "A Unified Algorithm for Elementary Functions," *Proc. 1971 Spring Joint Computer Conference*, pp. 379-385

[2] T.C. Chen, "Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots," *IBM J. of Research and Dev't.*, July 1972, pp. 380-388

[3] D. Cantor, G. Estrin, R. Turn, "Logarithmic and Exponential Function Evaluation in a Variable Structure Digital Computer," *IRE Trans. on Electronic Computers*, EC-11, April 1962, pp. 155-164

[4] W. Specker, "A Class of Algorithms for Ln x, Exp x, Sin x, Cos x, $Tan^{-1}x$ and $Cot^{-1}x$," *IRE Trans. on Electronic Computers*, EC-14, 1965, pp. 85-86

[5] H.M. Ahmed, M. Morf, D.T. Lee and P. Ang, "A VLSI Speech Analysis Chip Set Based on Square Root Normalized Ladder Forms," *Proc. 1981 Int'l. Conf. on Acoustics, Speech and Signal Processing*, Atlanta, GA, pp. 648-653

[6] A. Naseem, D. Fischer, "A Modified CORDIC Algorithm," *Proc. 1985 Symposim on Computer Arithmetic*

[7] H. M. Ahmed, K. Fu, "A VLSI Array CORDIC Architecture", *Proc. Int'l. Conf. on Acoustics, Speech and Signal Proc.*, May 1989, Glasgow, UK

[8] M. Farmwald, *On the Design of High Performance Digital Arithmetic Units*, " Ph.D Dissertation, Dept. of Computer Science, Stanford University, 1981

[9] H.M. Ahmed, *Signal Processing Algorithms and Architectures*, Ph.D Disseration, Information Systems Laboratory, Stanford University, June, 1982

[10] G. Haviland, A. Tuzynski, "A CORDIC Arithmetic Processor Chip," *IEEE Trans. on Computers*, Vol. C-29, No. 2, February, 1980

Appendix

Proof of Proposition 1:

Consider the final step of the hybrid algorithm to be a pure rotation given by:

$$X_n = \begin{bmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \end{bmatrix} X_k$$

Let $\cos\hat{\theta}_k$ denote the approximation of $\cos\theta_k$ and let $\sin\hat{\theta}_k$ be the approximation of $\sin\theta_k$. In this case:

$$\begin{cases} \cos\hat{\theta}_k = 1 \\ \sin\hat{\theta}_k = \theta_k \end{cases}$$

The error in the final rotation, denoted ε, is given by:

$$\varepsilon = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \end{bmatrix} = X_n - \hat{X}_n$$

$$= \begin{bmatrix} \cos\theta_k - \cos\hat{\theta}_k & -(\sin\theta_k - \sin\hat{\theta}_k) \\ \sin\theta_k - \sin\hat{\theta}_k & \cos\theta_k - \cos\hat{\theta}_k \end{bmatrix} X_k$$

We wish to select k such that $|\varepsilon_x| < 2^{-n+1}$ and $|\varepsilon_y| < 2^{-n+1}$. Consider ε_x and assume $x_k, y_k \in [-1, 1)$, which can be achieved through appropriate scaling. Then:

$$|\varepsilon_x| = \left| x_k(\cos\theta_k - \cos\hat{\theta}_k) - y_k(\sin\theta_k - \sin\hat{\theta}_k) \right|$$

$$\leq |x_k| \left| \cos\theta_k - \cos\hat{\theta}_k \right| + |y_k| \left| \sin\theta_k - \sin\hat{\theta}_k \right|$$

$$\leq \left| \cos\theta_k - \cos\hat{\theta}_k \right| + \left| \sin\theta_k - \sin\hat{\theta}_k \right|$$

A similar set of arguments establishes the same bound on $|\varepsilon_y|$. Therefore, in order to achieve $|\varepsilon_x| < 2^{-n+1}$, it is sufficient to guarantee

$$\text{(A1)} \quad \begin{cases} \left| \cos\theta_k - \cos\hat{\theta}_k \right| < 2^{-n} \\ \left| \sin\theta_k - \sin\hat{\theta}_k \right| < 2^{-n} \end{cases}$$

Since this will also assure that $|\varepsilon_y| < 2^{-n+1}$, we will only deal with $|\varepsilon_x|$ in the sequel. Consider the first part of (A1)

$$\left| \cos\theta_k - \cos\hat{\theta}_k \right| = \left| \sum_{n=1}^{\infty} \frac{(-1)^n \theta_k^{2n}}{(2n)!} \right|$$

$$\leq \sum_{n=1}^{\infty} \frac{|\theta_k|^{2n}}{(2n)!}$$

$$\leq \sum_{n=1}^{\infty} \left(\theta_k^2\right)^n = \frac{1}{1-\theta_k^2} - 1 = \frac{\theta_k^2}{1-\theta_k^2}$$

Therefore, applying constraint (A1) requires solving the inequality:

$$\frac{\theta_k^2}{1-\theta_k^2} < 2^{-n}$$

which has an approximate solution of $|\theta_k| < 2^{-[n/2]}$, where $[x]$ denotes the smallest integer exceeding x. We need not repeat the above calculations for the second part of (A1) because, in the region of interest,

$$\left| \cos\theta_k - \cos\hat{\theta}_k \right| \geq \left| \sin\theta_k - \sin\hat{\theta}_k \right|$$

Finally, applying lemma 1, yields $k > \left[\frac{n}{2}\right] + 1$.

Remarks:

(1) Although we have only considered the circular case, a similar bound can be derived for the hyperbolic functions.

(2) The reader may be concerned about the tightness of this bound given all the approximations we have made in the proof, i.e., can the actual value of k required be much smaller than the bound? In fact the bound is quite tight. This can be seen by trying to select k such that θ_k is small enough that the truncated terms of the Taylor series are *individually* smaller than can be accomodated within the n bit operand representation. Therefore, k must be chosen such that:

$$\theta_k < 1 \qquad \text{and} \qquad \frac{\theta_k^2}{2} < 2^{-n+1}$$

i.e:

$$|\theta_k| < 2^{\left(1-\frac{n}{2}\right)}$$

From lemma 1, this requires:

$$k > \left[\frac{n-2}{2}\right] + 1$$

$$= \left[\frac{n}{2}\right]$$

This is as well as we could hope to do because guaranteeing that the truncated series terms are all individually smaller than 2^{-n+1} does not guarantee that the Taylor remainder will also be smaller than this bound.

Table 2

Wordlength (bits)	$\frac{T_h}{T_m}$	$\frac{T_h}{T_c}$
24	2.63	.875
32	3.13	.781

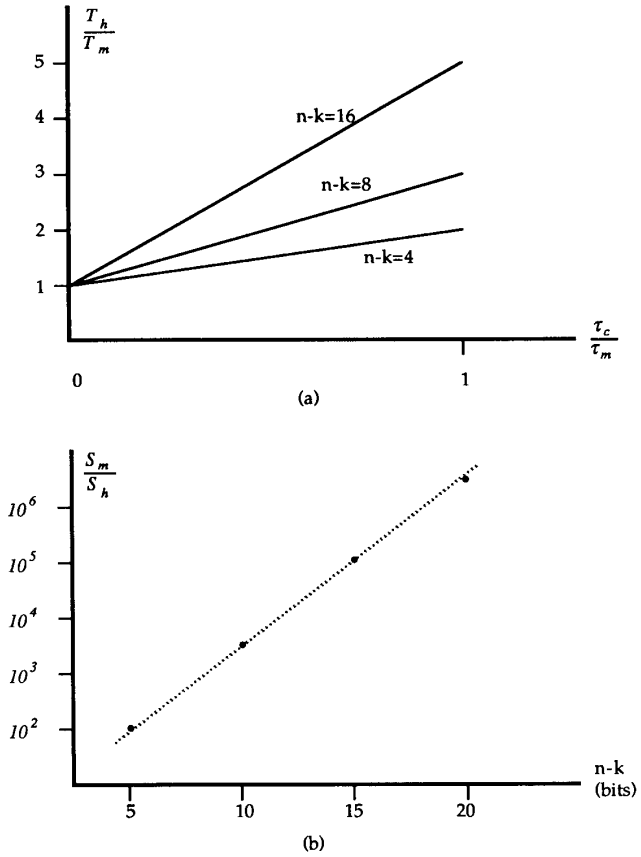

Figure 2

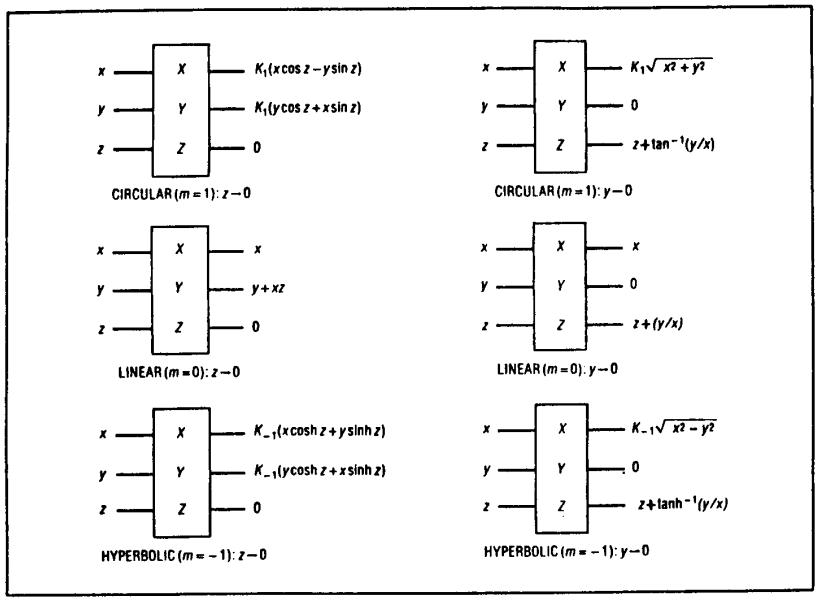

Figure 1

Table 1

Wordlength (bits)	$\frac{T_h}{T_m}$	$\frac{T_h}{T_c}$	$\frac{S_m}{S_h}$
24	2.5	.833	12870
32	3	.75	205900