Journal of VLSI Signal Processing, 11, 1–12 (1999)

The Symmetric Table Addition Method for Accurate Function Approximation

JAMES E. STINE AND MICHAEL J. SCHULTE
Computer Architecture and Arithmetic Laboratory
Electrical Engineering and Computer Science Department
Lehigh University
Bethlehem, PA 18015, USA

jes6, mschulte@eecs.lehigh.edu

Received February 22, 1998; Revised December 15, 1998

Editor: Valerie Taylor

Abstract.
This paper presents a high-speed method for computing elementary functions using parallel table lookups and multi-operand addition. Increasing the number of tables and inputs to the multi-operand adder significantly reduces the amount of memory required. Symmetry and leading zeros in the table coefficients are used to reduce the amount of memory even further. This method has a closed-form solution for the table entries and can be applied to any differentiable function. For 24-bit operands, this method requires two to three orders of magnitude less memory than conventional table lookups.

Keywords: Elementary functions, table lookups, approximations, multi-operand addition, computer arithmetic, hardware design.

1. Introduction

Elementary function approximations are important in scientific computing, computer graphics, and digital signal processing applications. In the systolic array implementation of Cholesky decomposition, presented in [1], 30% of the cells approximate reciprocals and square roots. Function approximations are also used extensively in single-value decomposition [2], vector normalization [3], artificial neural networks [4], signal generators [5], [6], [7], logarithmic number system processors [8], and 3D graphics applications [9].

Due to their frequent use in lighting calculations for 3D image processing, special instructions for approximating reciprocals and reciprocal roots are included in the the Motorola AltiVec [10] and Advanced Micro Devices 3DNow! [11] Instruction Set Extensions. Approximations for reciprocals, square roots, and reciprocal roots, are also needed for quadratically converging divide and square root algorithms [12], [13], [14], [15].

Software has been used successfully to evaluate elementary functions with techniques such as polynomial and rational approximations [16], [17]. However, since most software techniques require several iterations and have function call overhead, they are too slow for numerically intensive applications.

Dedicated hardware for elementary function approximations helps alleviate the speed limitations of software routines [18], [19]. Commercial processors, such as the AMD K5 [20], the Cryix 83D87 [21],

and the Intel Pentium [22] use table-driven methods for approximating elementary functions. Other hardware techniques include parallel polynomial approximations [23] [24], rational approximations [25], and shift-and-add algorithms, such as CORDIC [26], [27], [28].

With recent increases in transistor density [29] and improvements in memory technology [30], table-lookup methods for elementary functions are becoming more common. Previously, techniques were introduced that use two or more parallel tables followed by an addition to evaluate the elementary functions [31], [32]. These techniques provide high-speed approximations and require less memory than standard table lookups. A commercial implementation of these techniques is found in the AMD K-7 microprocessor, which uses two table lookups followed by an addition to approximate reciprocals and reciprocal roots [33]. These approximations are used to implement the floating point reciprocal and reciprocal root approximation instructions required by 3DNow! Technology [34].

Recently, the Symmetric Bipartite Table Method (SBTM) was introduced [35]. Similar to the techniques in [31], [32], and [33], this method employs an accurate approximation technique in which two parallel table lookups are followed by an addition. The SBTM, however, requires less memory than similar techniques because it takes advantage of symmetry and leading zeros in the table entries. Although a small amount of combinational logic is required to take advantage of symmetry, the reduction in memory size more than compensates for this. The SBTM also has a closed-form solution for the table entries and can be applied to any differentiable function.

This paper presents an extension to the SBTM called the Symmetric Table Addition Method (STAM), which uses more than two parallel table lookups followed by a multi-operand addition. Although the STAM requires more hardware than the SBTM, it provides a large reduction in memory size. The organization of this paper is as follows: Section 2 discusses function approximations using table addition methods. Section 3 gives a closed-form solution for calculating the table entries. Section 4 gives an error analysis of the SBTM and STAM and shows how to produce results that are accurate to within one unit in the last place (ulp). Section 5 illustrates how the SBTM and STAM use symmetry in the table entries to further reduce the memory requirements. Section 6 presents results using the STAM to reduce the memory requirements. Section 7 compares the STAM to previous table addition methods. Section 8 shows how the STAM can be modified to provide accurate initial approximations. Section 9 presents concluding remarks. The material presented in this paper is an extension of [36]. Software and additional material on the STAM is available at http://www.eecs.lehigh.edu/~caar/STAM.html.

2. Table Addition Approximations

To approximate a function using bipartite tables, the input operand is separated into three parts. The three partitions are denoted as x_0, x_1, and x_2 and have lengths of n_0, n_1, and n_2, respectively. The value of the input operand is $x = x_0 + x_1 + x_2$ and it has a length of $n = n_0 + n_1 + n_2$. The function is approximated as

$$f(x) \approx a_0(x_0, x_1) + a_1(x_0, x_2) \tag{1}$$

The $n_0 + n_1$ most significant bits of x are inputs to a table that provides the coefficient $a_0(x_0, x_1)$, and the n_0 most significant and n_2 least significant bits of x are inputs to a table that provides the coefficient $a_1(x_0, x_2)$. The outputs from the two tables are summed to produce an approximation $\widetilde{f(x)}$. This is shown in Figure 1, where the lengths of $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$ are denoted as p_0 and p_1, respectively.

This technique is extended by partitioning the input operand x into $m + 1$ parts, $x_0, x_1, \ldots, x_m$, with lengths of $n_0, n_1, \ldots, n_m$, respectively. This approximation takes the form

$$f(x) \approx \sum_{i=1}^{m} a_{i-1}(x_0, x_i) \tag{2}$$

The hardware implementation of this method has m parallel table lookups followed by an m-input multi-operand adder. The i^{th} table takes as inputs x_0 and x_i. The sum of the outputs from the tables produces an approximation to $f(x)$. By dividing the input operand into a larger number of partitions, smaller tables are needed. This method is illustrated in Figure 2, where the lengths of $a_0(x_0, x_1)$ through $a_{m-1}(x_0, x_m)$ are denoted as p_0 through p_{m-1}, respectively. The approximation to $f(x)$, denoted as $\widetilde{f(x)}$, has length p.

The block diagram in Figure 2 is easily implemented in hardware. The hardware has a one n-bit register and one p-bit register to store the input and out-

put operands, respectively. It also requires an m-input multi-operand adder to sum the table outputs. The total amount of memory for the table lookups is

$$\sum_{i=1}^{m} 2^{n_0+n_i} \cdot p_{i-1} \tag{3}$$

3. Selecting the Coefficients

This section presents the method for selecting the coefficients in each of the parallel tables for the SBTM and STAM. Initially, it is assumed that $0 \leq x < 1 - 2^{-n}$. The approximations for the SBTM and STAM are based on Taylor series expansions centered about the point $x_0 + x_1 + \delta_2$. The value

$$\delta_2 = 2^{-n_0-n_1-1} - 2^{-n_0-n_1-n_2-1} \tag{4}$$

is exactly halfway between the minimum and maximum values for x_2. Using the first two terms of the Taylor series results in the following approximation

$$f(x) \approx f(x_0 + x_1 + \delta_2) + f'(x_0 + x_1 + \delta_2)(x_2 - \delta_2) \tag{5}$$

As discussed in Section 2, the omission of the higher order terms in the Taylor series leads to a small approximation error. The first coefficient is selected as the first term in the Taylor series expansion.

$$a_0(x_0, x_1) = f(x_0 + x_1 + \delta_2) \tag{6}$$

Having the second coefficient depend on x_0, x_1, and x_2 would make the SBTM impractical due to its memory size. Therefore, the second coefficient is selected as

$$a_1(x_0, x_2) = f'(x_0 + \delta_1 + \delta_2)(x_2 - \delta_2) \tag{7}$$

This corresponds to the second term of the Taylor series with x_1 replaced by the constant δ_1, where

$$\delta_1 = 2^{-n_0-1} - 2^{-n_0-n_1-1} \tag{8}$$

is exactly halfway between the minimum and maximum values for x_1.

One benefit of this method is that the magnitude of the second coefficient is substantially less than the first coefficient, which allows the width of the second table to be reduced. Since $\mid x_2 - \delta_2 \mid < 2^{-n_0-n_1-1}$, the magnitude of the second coefficient is bounded by

$$\mid a_1(x_0, x_2) \mid < \mid f'(\xi_1) \mid 2^{-n_0-n_1-1} \tag{9}$$

where ξ_i is the point on $[0, 1)$ at which $\mid f^i(x) \mid$ takes its maximum value. This results in approximately

$$n_0 + n_1 + 1 + \log_2(\mid f(\xi_0)/f'(\xi_1) \mid) \tag{10}$$

leading zeros (or leading ones if $a_1(x_0, x_2) < 0$). These leading zeros (or ones) are not stored in memory, but are obtained by sign-extending the most significant bit of $a_1(x_0, x_2)$ before performing the carry propagate addition.

Similar to the SBTM, the coefficients for the STAM are generated so that they have a large number of leading zeros. Although, the STAM requires more tables

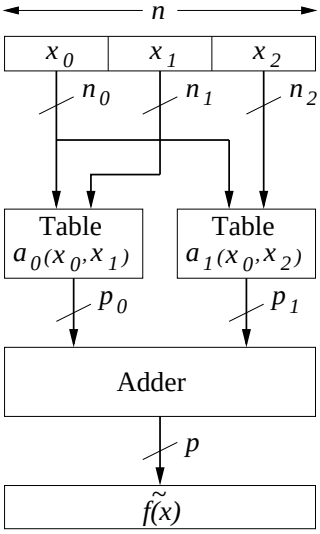

Fig. 1. Bipartite Table Method Block Diagram.

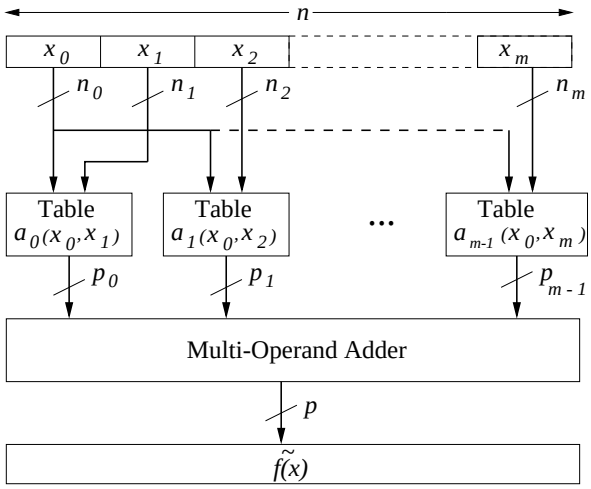

Fig. 2. Block Diagram for the Generalized Table Addition Method.

than the SBTM, the size of each table and the total memory size is reduced. The number of inputs to the adder, however, is increased.

It is again assumed that $0 \leq x \leq 1 - 2^n$, and thus

$$\begin{aligned} 0 \leq x_0 &\leq 1 - 2^{-n_0} \\ 0 \leq x_i &\leq 2^{-n_{0:i-1}} - 2^{-n_{0:i}} \quad (1 \leq i \leq m) \end{aligned} \tag{11}$$

where $n_{j:k} = \sum_{i=j}^{k} n_i$. To reduce the approximation error and create symmetry in the table coefficients, δ_i is defined to be exactly halfway between the minimum and maximum values of x_i, which gives

$$\delta_i = 2^{-n_{0:i-1}-1} - 2^{-n_{0:i}-1} \quad (1 \leq i \leq m) \tag{12}$$

It should be noted that δ_2 from the SBTM is equivalent to $\delta_{2:m}$ from the STAM, and x_2 from the SBTM is equivalent to $x_{2:m}$ from the STAM. The two term Taylor series expansion of $f(x)$ about $x_0 + x_1 + \delta_{2:m}$ is

$$\begin{aligned} f(x) &\approx f(x_0 + x_1 + \delta_{2:m}) \\ &+ f'(x_0 + x_1 + \delta_{2:m})(x_{2:m} - \delta_{2:m}) \end{aligned} \tag{13}$$

Similar to SBTM, x_1 is replaced by δ_1 which gives

$$\begin{aligned} f(x) &\approx f(x_0 + x_1 + \delta_{2:m}) \\ &+ f'(x_0 + \delta_1 + \delta_{2:m})(x_{2:m} - \delta_{2:m}) \end{aligned} \tag{14}$$

The second term in this approximation is then distributed into $m - 1$ terms, which gives

$$\begin{aligned} f(x) &\approx f(x_0 + x_1 + \delta_{2:m}) \\ &+ f'(x_0 + \delta_1 + \delta_{2:m}) \cdot \sum_{i=2}^{m} (x_i - \delta_i) \end{aligned} \tag{15}$$

Thus, the values for the coefficients are

$$\begin{aligned} a_0(x_0, x_1) &= f(x_0 + x_1 + \delta_{2:m}) \\ a_{i-1}(x_0, x_i) &= f'(x_0 + \delta_1 + \delta_{2:m})(x_i - \delta_i)(2 \leq i \leq m) \end{aligned} \tag{16}$$

The number of leading zeros (or ones) in $a_{i-1}(x_0, x_i)$ is determined by the bound

$$| a_{i-1}(x_0, x_i) | < | f'(\xi_1) | 2^{-n_{0:i}-1} \quad (2 \leq i \leq m) \tag{17}$$

4. Error Analysis

The SBTM and STAM both have errors due to

1. Approximating $f(x)$ with the first two terms of its Taylor series expansion
2. Replacing x_1 by δ_1 in the Taylor series expansion
3. Rounding the coefficients
4. Rounding the final result

By controlling each of these errors, the SBTM and STAM produce results that are faithfully rounded (i.e. the computed result differs from the true result by less than one unit in the last place (ulp) [14]).

Omitting the higher order terms in the Taylor series approximation results in an absolute error of

$$\begin{aligned} \epsilon_0 &= | \sum_{i=2}^{\infty} f^i(x_0 + x_1 + \delta_{2:m})(x_{2:m} - \delta_{2:m})^i | \\ &\approx | f''(x_0 + x_1 + \delta_{2:m}) | (x_{2:m} - \delta_{2:m})^2 \end{aligned} \tag{18}$$

Since $| x_{2:m} - \delta_{2:m} | < 2^{-n_0-n_1-1}$, this error is bounded by

$$\epsilon_0 < | f''(\xi_2) | 2^{-2n_0-2n_1-2} \tag{19}$$

Replacing x_1 by δ_1 in the Taylor series expansion results in an absolute error of

$$\begin{aligned} \epsilon_1 &= | f'(x_0 + x_1 + \delta_{2:m})(x_{2:m} - \delta_{2:m}) - \\ &\quad f'(x_0 + \delta_1 + \delta_{2:m})(x_{2:m} - \delta_{2:m}) | \\ &\approx | f''(x_0 + \delta_{2:m})(x_1 - \delta_1)(x_{2:m} - \delta_{2:m}) | \end{aligned} \tag{20}$$

Since $| x_1 - \delta_1 | < 2^{-n_0-1}$ and $| x_{2:m} - \delta_{2:m} | < 2^{-n_0-n_1-1}$, this error is bounded by

$$\epsilon_1 < | f''(\xi_2) | 2^{-2n_0-n_1-2} \tag{21}$$

To limit rounding error, the coefficients and the final result are rounded using a method similar to the one described in [14]. If the final result has p_f fraction bits, and the coefficients each have $(p_f + g)$ fraction bits, then rounding is performed as follows:

1. If m is even, $a_0(x_0, x_1)$ is rounded to the nearest number with $(p_f + g)$ fraction bits, and the $(p_f + g + 1)$-th fraction bit is set to zero. Otherwise, $a_0(x_0, x_1)$ is truncated to $(p_f + g)$ fraction bits, and the $(p_f + g + 1)$-th fraction bit is set to one.
2. For $2 \leq i \leq m$, $a_{i-1}(x_0, x_i)$ is truncated to $(p_f + g)$ fraction bits, and the $(p_f + g + 1)$-th fraction bit is set to one.
3. $\widetilde{f(x)}$ is rounded to the nearest number with p_f fraction bits.

This method guarantees that the maximum absolute error in rounding each coefficient is bounded by

2^{-p_f-g-1}, and that the least significant bit of their sum is always a one. The total error due to using m finite precision coefficients is bounded by

$$\epsilon_2 < m \times 2^{-p_f-g-1} \tag{22}$$

Since the least significant bit before rounding is guaranteed to be a one, the maximum absolute error in rounding the final result to p_f fraction bits using round-to-nearest is

$$\epsilon_3 \leq 2^{-p_f-1} - 2^{-p_f-g-1} \tag{23}$$

To ensure that each approximation is faithfully rounded, the sum of the four errors should be less than one unit in the last place, which gives the constraint

$$\epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon_3 < 2^{-p_f} \tag{24}$$

Replacing each error by its upper bound and combining terms gives the requirement

$$\underbrace{\mid f''(\xi_2) \mid 2^{-2n_0-n_1-2}(1+2^{n_1})}_{A} + \underbrace{(m-1) \times 2^{-p_f-g-1} + 2^{-p_f-1}}_{B} < 2^{-p_f} \tag{25}$$

The terms labeled as A and B corresponds to approximation errors, $\epsilon_0 + \epsilon_1$, and rounding errors, $\epsilon_2 + \epsilon_3$, respectively. Equation (24) is satisfied and faithful rounding is guaranteed if the following two conditions are met

$$\begin{aligned} 2n_0 + n_1 &\geq p_f + \log_2(\mid f''(\xi_2) \mid) \\ g &\geq 2 + \log_2(m-1) \end{aligned} \tag{26}$$

For example, assume the goal is to approximate the sine function with the SBTM to $p_f = 12$ fraction bits on $[0, 1)$. If x is partitioned with $n_0 = 4$, $n_1 = 4$, and $n_2 = 4$, then $8 + 4 \geq 12 + \lceil \log_2(1/\sqrt{2}) \rceil$ satisfies the first inequality in Equation (25). Providing two guard bits, satisfies the second inequality in Equation (25). Figure 3, shows the error plot for this approximation and demonstrates that faithful rounding is achieved.

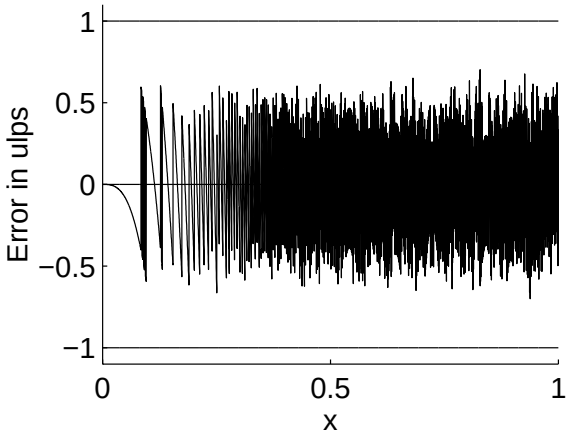

Fig. 3. Error for a 12-Bit SBTM Approximation of Sin(x).

5. Symmetry and Hardware Designs

The table for $a_i(x_0, x_i)$ has $2^{n_0+n_i}$ words. The method for selecting the coefficients causes tables $a_1(x_0, x_2)$ through $a_{m-1}(x_0, x_m)$ to be symmetric and allows their sizes to be reduced to $2^{n_0+n_i-1}$ words. The symmetry achieved in the SBTM and the STAM is obtained because

- $2\delta_i - x_i$ is the one's complement of x_i
- $a_{i-1}(x_0, 2\delta_i - x_i)$ is the one's complement of $a_{i-1}(x_0, x_i)$

These properties are demonstrated in Table 1 which gives eight table entries for $f(x) = cos(x)$ when $m = 2$, $n_0 = 2$, $n_1 = 2$, $n_2 = 3$, $g = 2$, and $p = 7$. The symmetry can be easily seen in Table 1 for different values of x.

Each value for x_2 in the first half of the table has a one's complement in the second half of the table, as shown by the bold-faced binary values. The corresponding table entries for $a_1(x_0, x_2)$ also have one's complements, shown in bold. There is no need to store the bits of $a_1(x_0, x_2)$ which are not bold-faced, since these correspond to leading zeros (or ones) that can be obtained by sign-extension.

With the SBTM the $a_1(x_0, x_2)$ table is folded by examining the most significant bit of x_2. If this bit is a zero, then the remaining bits of x_2 remain unchanged, and the value is read from the $a_1(x_0, x_2)$ table and added to the $a_0(x_0, x_1)$ table. However, if this bit is a one, then the remaining bits of x_2 are complemented,

Table 1. Table Entries for $a_1(x_0, x_2)$.

x		$a_1(x_0, x_2)$	
decimal	binary	decimal	binary
0.500000	0.1000**000**	+0.0166016	0.00000**1000**1
0.507812	0.1000**001**	+0.0107422	0.00000**0101**1
0.515625	0.1000**010**	+0.0068359	0.00000**0011**1
0.523438	0.1000**011**	+0.0029297	0.00000**0001**1
0.531250	0.1000**100**	-0.0029297	1.11111**1110**1
0.539062	0.1000**101**	-0.0068359	1.11111**1100**1
0.546875	0.1000**110**	-0.0107422	1.11111**1010**1
0.554688	0.1000**111**	-0.0166016	1.11111**0111**1

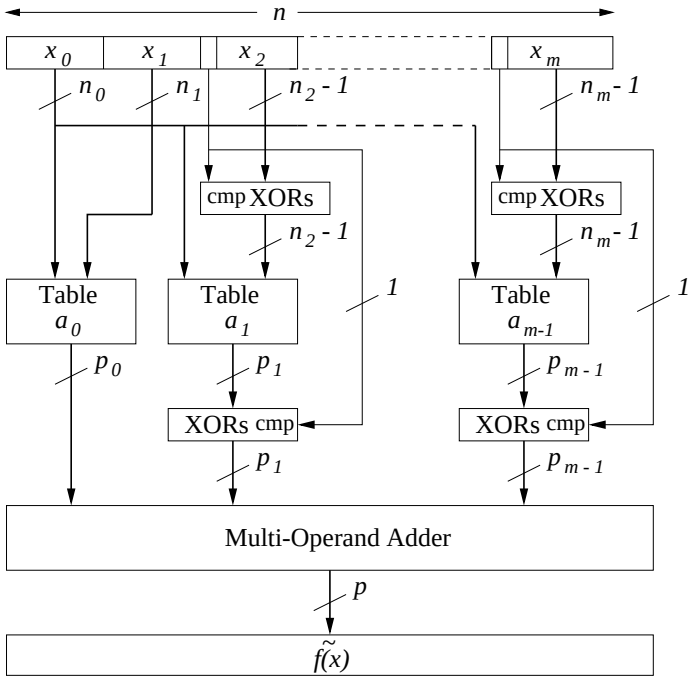

Fig. 4. Symmetric Table Addition Method Block Diagram.

and used to address the a_1 table. The output is then complemented and added to $a_0(x_0, x_1)$. This method is extended for the STAM by examining the most significant bit of x_i ($2 \leq i \leq m$), as shown in Figure 4.

If the most significant bit of x_i is a one, a row of $n_i - 1$ XOR gates complements the remaining bits of x_i which addresses the $a_{i-1}(x_0, x_i)$ table, and the output of the table is complemented using a row of p_i exclusive-or gates. The most-significant bits and the least-significant bit of the table do not need to be stored, since these bits are known in advance.

In [37], evaluation of squares is achieved using a similar approach. Unlike their technique, which only applies to integer square, the SBTM and the STAM can be applied to any differentiable function.

6. Results

In the previous section, it is initially assumed that the $0 \leq x < 1$. However, the SBTM and the STAM can be applied to different intervals. For example, reciprocals are often calculated for $1 \leq x < 2$. Thus, the most-significant bit of x is known to be a one and is not used in the table lookup. Table 2 shows the input and output ranges for the elementary functions discussed in the remainder of this paper.

Tables 3 and 4 display optimal partitions and the corresponding table dimensions for various elementary function approximations using 16 and 24 bit operands. These partitions are chosen to minimize the total amount of memory while still satisfying the constraints given in Equation (25). As the number of

Table 2. Ranges and Derivative Values for Selected Functions.

$f(x)$	Input	Output	$\mid f'(\xi_1) \mid$	$\mid f''(\xi_2) \mid$
$1/x$	[1, 2)	(0.5, 1]	1	2
$\sqrt{x}$	[1, 2)	[1, $\sqrt{2}$)	1/2	1/4
$\sin(x)$	[0, 1)	[0, $1/\sqrt{2}$)	1	$1/\sqrt{2}$
2^x	[0, 1)	[1, 2)	$2\ln 2$	$2(\ln(2))^2$
$\log_2(x)$	[1, 2)	[0, 1)	$\log_2(e)$	$\log_2(e)$
$\ln(x)$	[1, 2)	[0, $\ln(2)$)	1	1
$1/\sqrt{x}$	[1, 2)	($1/\sqrt{2}$,1]	1/2	3/4

Table 3. Bit Partitions and Table Sizes for 16-Bit Operands.

$f(x)$	Tables	Bit Partitions	Table Sizes	Total Memory
$1/x$	1	15	$2^{15} \cdot 15$	491,520
$1/x$	2	$6, 4, 5$	$2^{10}(17 + 7)$	24,576
$1/x$	3	$7, 2, 3, 3$	$2^{9}(18 + 9 + 6)$	16,896
$1/x$	4	$7, 2, 2, 2, 2$	$2^{9} \cdot 19 + 2^{8}(10 + 8 + 6)$	15,872
$\sqrt{x}$	1	15	$2^{15} \cdot 15$	491,520
$\sqrt{x}$	2	$4, 5, 6$	$2^{9}(17 + 7)$	12,288
$\sqrt{x}$	3	$5, 3, 3, 4$	$2^{8} \cdot 18 + 2^{7} \cdot 9 + 2^{8} \cdot 6$	7,296
$\sqrt{x}$	4	$5, 3, 2, 2, 3$	$2^{8} \cdot 19 + 2^{6}(10 + 8) + 2^{7} \cdot 6$	6,784
$\sin(x)$	1	16	$2^{16} \cdot 16$	1,048,576
$\sin(x)$	2	$6, 4, 6$	$2^{10} \cdot 18 + 2^{11} \cdot 7$	32,768
$\sin(x)$	3	$7, 2, 3, 4$	$2^{9}(19 + 9) + 2^{10} \cdot 6$	20,480
$\sin(x)$	4	$7, 2, 2, 2, 3$	$2^{9} \cdot 20 + 2^{8}(10 + 8) + 2^{9} \cdot 6$	17,920
2^x	1	16	$2^{16} \cdot 15$	983,040
2^x	2	$5, 5, 6$	$2^{10}(17 + 7)$	24,576
2^x	3	$6, 3, 3, 4$	$2^{9} \cdot 18 + 2^{8} \cdot 9 + 2^{9} \cdot 6$	14,592
2^x	4	$6, 3, 2, 2, 3$	$2^{9} \cdot 19 + 2^{7}(10 + 8) + 2^{8} \cdot 6$	13,568

Table 4. Bit Partitions and Table Sizes for 24-Bit Operands.

$f(x)$	Tables	Bit Partitions	Table Sizes	Total Memory
$1/x$	1	23	$2^{23} \cdot 23$	192,937,984
$1/x$	2	$9, 7, 7$	$2^{16} \cdot 25 + 2^{15} \cdot 9$	1,933,312
$1/x$	3	$11, 3, 4, 5$	$2^{14}(26 + 12) + 2^{15} \cdot 8$	884,736
$1/x$	4	$11, 3, 3, 3, 3$	$2^{14} \cdot 27 + 2^{13}(13 + 10 + 7)$	688,128
$1/x$	5	$11, 3, 2, 2, 2, 3$	$2^{14} \cdot 27 + 2^{12}(13 + 11 + 9) + 2^{13} \cdot 7$	634,880
$1/x$	6	$11, 3, 1, 2, 2, 2, 2$	$2^{14} \cdot 28 + 2^{11} \cdot 14 + 2^{12}(13 + 11 + 9 + 7)$	651,264
$\sqrt{x}$	1	23	$2^{23} \cdot 23$	192,937,984
$\sqrt{x}$	2	$7, 7, 9$	$2^{14} \cdot 25 + 2^{15} \cdot 10$	737,280
$\sqrt{x}$	3	$8, 5, 5, 5$	$2^{13} \cdot 26 + 2^{12}(12 + 7)$	290,816
$\sqrt{x}$	4	$9, 3, 3, 4, 4$	$2^{12} \cdot 27 + 2^{11} \cdot 14 + 2^{12}(11 + 7)$	212,992
$\sqrt{x}$	5	$9, 3, 2, 3, 3, 3$	$2^{12} \cdot 27 + 2^{10} \cdot 14 + 2^{11}(12 + 9 + 6)$	180,224
$\sqrt{x}$	6	$9, 3, 2, 2, 2, 2, 3$	$2^{12} \cdot 28 + 2^{10}(15 + 13 + 11 + 9) + 2^{11} \cdot 7$	178,176
$\sin(x)$	1	24	$2^{24} \cdot 24$	402,653,184
$\sin(x)$	2	$8, 8, 8$	$2^{16} \cdot 26 + 2^{15} \cdot 9$	1,998,848
$\sin(x)$	3	$10, 4, 5, 5$	$2^{14} \cdot (27 + 12 + 7)$	753,664
$\sin(x)$	4	$10, 4, 3, 3, 4$	$2^{14} \cdot 28 + 2^{12}(13 + 10) + 2^{13} \cdot 7$	610,304
$\sin(x)$	5	$11, 2, 2, 3, 3, 3$	$2^{13} \cdot 28 + 2^{12} \cdot 14 + 2^{13}(12 + 9 + 6)$	507,904
$\sin(x)$	6	$11, 2, 2, 2, 2, 2, 3$	$2^{13} \cdot 29 + 2^{12}(15 + 13 + 11 + 9) + 2^{13} \cdot 7$	491,520
2^x	1	24	$2^{24} \cdot 23$	385,875,968
2^x	2	$8, 7, 9$	$2^{15} \cdot 25 + 2^{16} \cdot 10$	1,474,560
2^x	3	$9, 5, 5, 5$	$2^{14} \cdot 26 + 2^{13}(12 + 7)$	581,632
2^x	4	$10, 3, 3, 4, 4$	$2^{13} \cdot 27 + 2^{12} \cdot 14 + 2^{13}(11 + 7)$	425,984
2^x	5	$10, 3, 2, 3, 3, 3$	$2^{13} \cdot 27 + 2^{11} \cdot 14 + 2^{12}(12 + 9 + 6)$	360,448
2^x	6	$10, 3, 2, 2, 2, 2, 3$	$2^{13} \cdot 28 + 2^{11}(15 + 13 + 11 + 9) + 2^{12} \cdot 7$	356,352

tables increases, the total memory requirements decrease.

Figures 5 and 6 show the compression achieved with the optimized partitioning as the number of tables varies. Compression is defined as the amount of memory required by a standard table lookup divided

by the amount of memory required by the method being examined [14]. For 16-bit operands, the compression ratios range from 20 to 72. For 24-bit operands, they range from 100 to 1083.

Delay and area estimates were made to compare standard table lookups to SBTM and STAM implementations. The SBTM uses a carry-lookahead adder to sum the outputs of the two tables. For the STAM implementations, the multi-operand adder consists of a tree of carry-save adders followed by a carry-lookahead adder. All table lookups are implemented using read only memories (ROMs). Each implementation uses the LCA300K ASIC standard-cell library from LSI Logic, which uses a 0.6-micron CMOS technology with two layers of metal. The nominal operating voltage for this library is 5.0 Volts at 25° C. The Leonardo/Spectrum Synthesis Tools from Exemplar were used to generate the designs. The synthesis tools accept a VHDL description as input, and uses gate and wire load models to estimate the delay. The area estimates are given in number of equivalent gates.

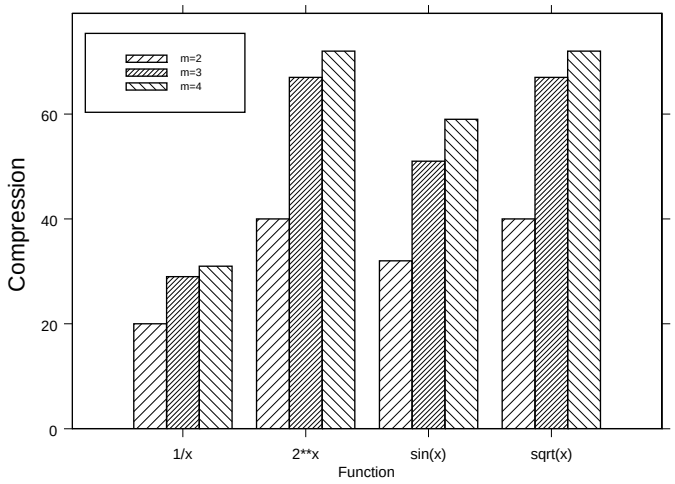

Fig. 5. STAM Compression Ratios for 16-Bit Operands.

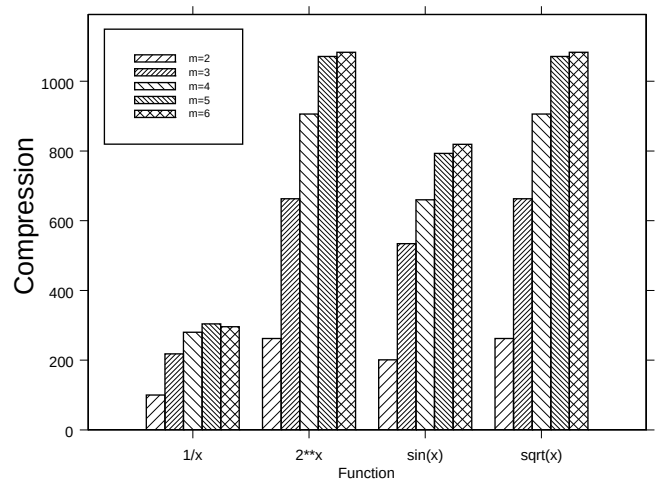

Fig. 6. STAM Compression Ratios for 24-Bit Operands.

Table 5 gives pre-layout area and delay estimates for implementations that approximate $1/x$ for 16-bit operands. To provide a better understanding of the tradeoffs between the ROM table lookups and the combinational logic, the area and delay for each of these is given separately. Although the standard table implementation is 6% faster than the STAM with $m = 3$, it requires more than 25 times as much area. As the number of tables increases, the area and delay of the ROMs decreases, but the area and delay of the combinational logic increases. When going from a standard table lookup ($m = 1$) to the SBTM ($m = 2$), there is a large decrease in both the area and delay of the ROMs. The additional delay due to the XOR gates and carry-lookahead adder, however, result in an overall increase in delay. When going from $m = 2$ to $m = 3$ there is a still a fairly significant decrease in area, since the size of the ROMs, which dominates the area, decreases by about 27%. The total delay also decreases slightly because the decrease in the delay from smaller ROMs is greater than the increase in delay from the combinational logic, which is equivalent to a carry save adder delay. When going from $m = 3$ to $m = 4$, the area again decreases, due to an decrease in the total size of the ROMs. However, the worst case delay through the ROMs does not decrease, and thus an overall increase in delay results. For 16-bit operands little or no benefit is realized by increasing the number of tables beyond $m = 4$.

7. Comparison To Previous Methods

In [31], the 3-block method, which uses two table lookups followed by addition, is presented. With this method, the elementary functions are approximated based on the partial product arrays of their Taylor se-

Table 5. Area and Delay Estimates for $1/x$ with 16-Bit Operands.

	Area (equivalent gates)			Delay (ns)		
m	ROM	Logic	Total	ROM	Logic	Total
1	268,319	–	268,319	8.68	–	8.68
2	13,631	361	13,992	4.92	4.31	9.23
3	9,885	598	10,483	3.96	5.25	9.21
4	8,005	839	8,844	3.96	6.04	10.00

Table 6. The SBTM and the 3-Block Method for 24-Bit Operands.

	Symmetric Bipartite Tables		Table Sizes	
$f(x)$	n_0, n_1, n_2	Table Sizes	From [31]	Compression
$1/x$	10, 7, 7	$2^{16}l + 2^{15}(l-16)$	$2^{17}l + 2^{16}l$	2.5
$\ln(x)$	9, 6, 8	$2^{15}l + 2^{15}(l-16)$	$2^{16}l + 2^{16}l$	2.8
$\log_2(x)$	10, 6, 8	$2^{15}l + 2^{16}(l-15)$	$2^{17}l + 2^{15}l$	2.6
2^x	8, 7, 9	$2^{14}l + 2^{15}(l-15)$	$2^{17}l + 2^{15}l$	5.2

Table 7. The STAM and TLAM for $\ln(x)$ with 24-Bit Operands.

	STAM		Table Sizes	
Tables	Bit Partitions	Table Sizes	From [31]	Compression
2	9, 6, 8	$2^{15}l + 2^{16}(l-16)$	$2^{16}l + 2^{16}l$	2.2
3	10, 4, 4, 5	$2^{14}(2l-19) + 2^{13}(l-15)$	$2^{16}l + 2^{11}l$	2.7
4	10, 4, 3, 3, 3	$2^{14}l + 2^{12}(3l-54)$	$2^{16}l + 2^{8}l$	3.2
5	11, 2, 2, 2, 3, 3	$2^{13}(3l-39) + 2^{12}(2l-30)$	$2^{14}(6l)$	5.8

ries expansions. In terms of hardware complexity, the 3-block method is similar to the SBTM, however, the SBTM requires smaller tables. The 3-block method is used to approximate $1/x$, $\ln(x)$, $\log_2(x)$ and 2^x for 24-bit operands, with $0.5 \leq x < 1$. In Table 6, the memory requirements for the 3-block method and the SBTM are shown. To compare the two methods, Table 6 uses the notation presented in [31]. The value l denotes the number of bits in the output operand, plus a designated number of guard digits (i.e., $l = p + g$). The compression achieved by the SBTM compared to the method presented in [31] is given, assuming $l = 28$. The SBTM requires 2.5 to 5.2 times less memory than the 3-block method.

An extension to the 3-block method, referred to as the Table Look-up and Addition Method (TLAM), uses more than two tables [31]. Table 7 compares the STAM to the TLAM for $\ln(x)$, and shows that the STAM requires 2.2 to 5.8 times less memory than the TLAM. The STAM requires less mmemory because it

1. Employs a more accurate method to select the coefficients
2. Does not store the leading zeros (or ones) of $a_{i-1}(x_0, x_i)$
3. Takes advantage of symmetry in $a_{i-1}(x_0, x_i)$

8. STAM Initial Approximations

Initial approximations to reciprocals, square roots, and reciprocal roots are tyically used for quadratically converging algorithms, such as those described in [12], [13], [14], [15]. For these approximations, only the most significant bits of x are used to approximate $f(x)$. This allows smaller tables to be used, but results in additional error due to leaving off the least significant bits of x.

A modified version of the STAM, which uses only the most significant bits of x as inputs to the lookup tables, provides accurate initial approximations. With the modified STAM, x is divided into $m + 2$ partitions, x_0, x_1, ..., x_{m+1}. The $m + 1$ most significant partitions, x_0, x_1, $\ldots x_m$, are used as inputs to the lookup tables and the remaining bits, which correspond to x_{m+1} are not used. Typically, x_{m+1} is much larger than the other x_i.

To compensate for leaving of the bits of x_{m+1}, these bits are approximated as

$$\delta_{m+1} = 2^{-n_{0:m}-1} - 2^{-n_{0:m+1}-1} \tag{27}$$

which is halfway between the minimum and maximum values of x_{m+1}. The values for the coefficients are

$$\begin{aligned} a_0(x_0, x_1) &= f(x_0 + x_1 + \delta_{2:m+1}) \\ a_{i-1}(x_0, x_i) &= f'(x_0 + \delta_1 + \delta_{2:m+1})(x_i - \delta_i) \end{aligned} \tag{28}$$

Table 8. Partitions and Table Sizes for 16-Bit Initial Approximations.

$f(x)$	Tables	Bit Partitions	Table Sizes	Total Memory	Compression
$1/x$	1	16	$2^{16} \cdot 15$	983,040	1
$1/x$	2	$6, 5, 5$	$2^{11} \cdot 18 + 2^{10} \cdot 7$	44,032	22
$1/x$	3	$7, 3, 3, 3$	$2^{10} \cdot 19 + 2^{9} \cdot (9 + 6)$	27,136	36
$1/x$	4	$7, 3, 2, 2, 2$	$2^{10} \cdot 20 + 2^{8} \cdot (10 + 8 + 6)$	26,624	37
$\sqrt{x}$	1	15	$2^{15} \cdot 15$	491,520	1
$\sqrt{x}$	2	$5, 4, 6$	$2^{9} \cdot 19 + 2^{10} \cdot 9$	18,944	26
$\sqrt{x}$	3	$6, 2, 3, 4$	$2^{8} \cdot (20 + 11) + 2^{9} \cdot 8$	12,032	41
$\sqrt{x}$	4	$6, 2, 2, 2, 3$	$2^{8} \cdot 21 + 2^{7} \cdot (12 + 10) + 2^{8} \cdot 8$	10,240	48
$1/\sqrt{x}$	1	15	$2^{15} \cdot 15$	491,520	1
$1/\sqrt{x}$	2	$6, 4, 5$	$2^{10} \cdot (17 + 7)$	24,576	20
$1/\sqrt{x}$	3	$7, 2, 3, 3$	$2^{9} \cdot (18 + 9 + 6)$	16,896	29
$1/\sqrt{x}$	4	$7, 2, 2, 2, 2$	$2^{9} \cdot 19 + 2^{8} \cdot (10 + 8 + 6)$	15,872	31

Replacing x_{m+1} by δ_{m+1} in the approximation results in an absolute error of

$$\begin{aligned} \epsilon_4 &= \mid f(x_{0:m} + x_{m+1}) - f(x_{0:m} + \delta_{m+1}) \mid \\ &< \mid f'(\xi_1) \mid 2^{-n_{0:m}-1} \end{aligned} \tag{29}$$

Since initial approximations are intermediate values, it is typically not required that their maximum absolute error be less than one ulp. Therefore, it is not necessary to round the sum of the coefficients, which eliminates the rounding error ϵ_3. The maximum absolute error in the modified STAM is bounded by

$$\begin{aligned} &\epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon_4 < \\ &\mid f''(\xi_2) \mid 2^{-2n_0-n_1-2}(1 + 2^{-n_1}) + \\ &\mid f'(\xi_1) \mid 2^{-n_{0:m}-1} + m \cdot 2^{-p_f-g-1} \end{aligned} \tag{30}$$

The values for m, $p_f + g$, and n_i are selected to meet specified accuracy requirements.

Table 8 shows partitions and table sizes that result in a maximum absolute error that is less than 2^{-16} for $1/x$, $\sqrt{x}$, and $1/\sqrt{x}$. Compared to standard table lookups the STAM with $m = 4$ requires 37 to 48 times less memory. The $\sqrt{x}$ has the largest compression and $1/x$ has the smallest.

9. Conclusions

This paper presents the Symmetric Table Addition Method for approximating elementary functions. Compared to standard table lookups the memory requirements are reduced dramatically. The STAM also requires less memory than previous table addition methods because the tables are symmetric and have a large number of leading zeros. This method has simple hardware requirements and can be applied to a wide range of functions. It is suitable for applications that require high-speed elementary function approximations at low to moderate precisions (i.e. 24 bits or less). It can also be used to obtain accurate initial approximations, which are needed for quadratically converging divide and square root algorithms.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. MIP-9703421.

References

1. V. K. Jain, G. E. Perez, and J. M. Wills, "Novel Reciprocal and Square-Root VLSI Cell: Architecture and Application to Signal Processing," in *International Conference on Acoustics, Speech and Signal Processing*, vol. 2, pp. 1201–1204, 1991.
2. V. K. Jain and L. Lin, "Square-Root, Reciprocal, Sine/Cosine, Arctangent cell for Signal and Image Processing," in *International Conference on Acoustics, Speech and Signal Processing*, vol. 2, pp. 521–524, 1994.
3. H. Kwan, R. L. Nelson, and E. E. Swartzlander, Jr., "Cascaded Implementation of an Iterative Inverse-Square-Root Algorithm with Overflow Lookahead," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 114–123, 1995.
4. M. Zhang, S. Vassiliadis, and J. G. Delgado-Frias, "Sigmoid Generators for Neural Computing Using Piecewise Approximations," *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 1045–1050, 1996.
5. S. Mehrgardt, "Noise Spectra of Digital Sine-Generators Using the Table-lookup Method," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 31, no. 4, pp. 1037–1039, 1983.
6. J. A. McIntosh and E. E. Swartzlander, Jr., "High-speed Cosine Generator," in *Conference Record of the Twenty-Eighth Asilomar Conference on Signals, Systems and Computers*, pp. 273–277, 1994.
7. J. McClellan, R. Schafer, and M. Yoder, *DSP First: A Multimedia Approach*. Prentice Hall, 1998.
8. D. M. Lewis, "114 MFLOPS Logarithmic Number System Arithmetic Unit for DSP Applications," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 12, pp. 1547–1553, 1995.

9. H.-C. Shin, J.-A. Lee, and L.-S. Kim, "A Minimized Hardware Architecture of Fast Phong Shader Using Taylor Series Approximation in 3D Graphics," in *Proceedings of the International Conference on Computer Design. VLSI in Computers and Processors*, pp. 286–291, 1998.
10. *AltiVec Technology Programming Environments Manual*. Motorola, Inc., 1998.
11. *3DNow! Technology Manual*. November 1998. Available at http://www.amd.com/K6/k6docs/.
12. M. J. Flynn, "On Division by Functional Iteration," *IEEE Transactions on Computer*, vol. C-19, pp. 702–706, 1970.
13. S. F. Oberman and M. J. Flynn, "Division Algorithms and Implementations," *IEEE Transactions on Computer*, vol. C-46, pp. 833–854, 1997.
14. D. D. Sarma and D. W. Matula, ""Measuring the Accuracy of ROM Reciprocal Tables," in *Proceedings of the 11th Symposium on Computer Arithmetic*, pp. 95–102, July 1993.
15. C. V. Ramamoorthy and J. Goodman, "Some properties of Iterative Square-Rooting Methods Using High-Speed Multiplication," *IEEE Transactions on Computer*, vol. C-21, pp. 837–847, 1972.
16. W. Cody and W. Waite, *Software Manual for the Elementary Functions*. Prentice-Hall, 1980.
17. J. F. Hart, *et. al.*, *Computer Approximations*. Wiley, 1968.
18. J. S. Walther, "A Unified Approach for Elementary Functions," in *Spring Joint Computer Conference*, pp. 379–385, 1971.
19. J. M. Muller, *Elementary Function, Algorithms and Implementation*. Birkhauser Boston, 1997.
20. T. Lynch, A. Ahmed, M. J. Schulte, T. Callaway, , and R. Tisdale, "The K5 Transcendental Functions," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 222–229, 1995.
21. W. Ferguson and T. Brightman, "Accurate and Montone Approximations of some Transcendental Functions," in *Proceedings of the 10th Symposium on Computer Arithmetic*, pp. 237–244, 1991.
22. *Intel Pentium Processor Data Book*. Intel Corporation, 1994.
23. M. J. Schulte and E. E. Swartzlander, Jr, "Hardware Designs for Exactly Rounded Elementary Functions," *IEEE Transactions on Computer, Special Issue on Computer Arithmetic*, vol. C-44, pp. 964–973, 1994.
24. P. Farmwald, "High Bandwidth Evaluation of Elementary Functions," in *Proceedings of the 5th Symposium on Computer Arithmetic*, pp. 139–142, 1981.
25. I. Koren, *Computer Arithmetic and Algorithms*. Prentice Hall, 1993.
26. J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, vol. EC-8, pp. 330–334, 1959.
27. G. H. Haviland and A. A. Tuszynski, "A CORDIC arithmetic processor," *IEEE Transactions on Computers*, vol. C-29, pp. 68–79, 1980.
28. S. F. Hsiao and J. M. Delosme, "Householder CORIDC algorithms," *IEEE Transactions on Computers*, vol. C-44, pp. 990–1000, 1995.
29. Sematech, Inc., *The National Technology Roadmap for Semiconductors*, 1997.
    Available at http://notes.sematech.org/˜ 97pelec.htm.
30. A. R. Pelella, L. Pong-Fei, Y. H. Chan, W. V. Chan, W. V. Huott, U. Bakhru, S. Kowalczyk, P. Patel, J. Rawlins, and P. T. Wu, "A 2-nanosecond access, 500 MHz, 288Kb SRAM Macro," in *Symposium on VLSI Circuits. Digest of Technical Papers*, pp. 309–318, 1995.
31. H. Hassler and N. Takagi, "Function Evaluation by Table Look-up and Addition," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 10–16, 1995.
32. D. D. Sarma and D. W. Matula, "Faithful Bipartite Rom Reciprocal Tables," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 17–29, 1995.
33. S. F. Oberman, "Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7 Microprocessor," in *Proceedings of the 14th Symposium on Computer Arithmetic*, April 1999. (in press).
34. H. Kalish (editor) and J. Isaac (editor), *The AMD-K6 3D Processor*. Abacus Software, 1998.
35. M. J. Schulte and J. E. Stine, "Symmetric Bipartite Tables for Accurate Function Approximation," in *Proceedings of the 13th Symposium on Computer Arithmetic)*, pp. 175–183, 1997.
36. M. J. Schulte and J. E. Stine, "Accurate Function Approximations by Symmetric Table Lookup and Addition," in *11th International Conference on Application-Specific Systems, Architectures, and Processors)*, pp. 144–153, 1997.
37. C. Wey, "On Design of Efficient Square Generator," in *Proceedings International Conference on Computer VLSI Design in Computers and Processors)*, pp. 506–511, 1996.

James E. Stine is a Ph.D. candidate in electrical engineering at Lehigh University in Bethlehem, Pennsylvania. His reseach interests include virtually all apsects of the design of digital circuits and systems, with special focus on the exploration of computer arithmetic algorithms, computer architectures, and implementation of signal processing systems and the related design methodologies.

Michael J. Schulte is an assistant professor at Lehigh University. His teaching and research interests include computer arithmetic, computer architecture, application-specific processors, and VLSI design He has over 30 publications in these and related areas. His paper entitled "A Software Interface and Hardware Design for Variable-Precision Interval Arithmetic," won the 1995 Best Paper Award for the Journal of Reliable Computing.

Michael obtained the Ph.D. and M.S. degrees in Electrical and Computer Engineering from the University of Texas at Austin in 1996 and 1992, respectively. He received the B.S. degree in Electrical and Computer Engineering, with a second major in Computer Science, from the University of Wisconsin-Madison in 1991. While attending graduate school, he was supported by the Microelectronic and Computer Development (MCD) Fellowship, the Basdall Garder MCD Fellowship, and the TRW Graduate Fellowship.

Michael has worked as a Design Engineer for Crystal Semiconductor, Advanced Micro Devices, and Ross Technology, and has consulted for Centaur Technology and Analog Devices, Inc. He currently serves as a reviewer for the National Science Foundation, IEEE Transactions on Computers, the Journal of Microcomputer Systems Architecture, the Journal of Reliable Computing, and several international conference. He is the Branch Counselor for the student chapter of the IEEE at Lehigh University, and a member of the IEEE Computer Society.