Faithful Interpolation in Reciprocal Tables

Debjit Das Sarma
Texas Instruments
Dallas, Texas 75243

David W. Matula
Southern Methodist University
Dallas, Texas 75275

Abstract

We describe a table compression method employing finite precision linear interpolation in reciprocal tables. The interpolation method employs a compressed look-up table and a small sized multiplier to yield an output reciprocal as a simple direct operation. The leading bits of the arbitrarily precise input are used to index the table and a limited number of succeeding fractional bits are used to interpolate on the table employing a multiply/add operation. The low order bits of the product are rounded off so the output reciprocals are guaranteed correct to a unit in the last place, and provide a round-to-nearest reciprocal for over 90% of arbitrarily precise input arguments. The interpolation method generates $2k$-bit faithful reciprocals employing a k-bits-in $2k+2$-bits-out table and a $(k+3) \times (k+3)$ bit multiplier. A single precision faithful reciprocal can be generated employing a table of size 13 Kbytes and a 15×15 bit multiplier, compared to a table size of 46 Mbytes for conventional reciprocal tables. The table and dedicated small multiplier efficiently characterize a functional reciprocator unit with at most a couple of cycle latency.

1. Introduction and Summary

With the density of transistors that can be realized in an integrated circuit on a single chip rising so rapidly, it has become increasingly common for arithmetic circuits to include a reciprocal table to either assist or replace the division instruction. For low precision arithmetic computation, direct use of a suitably large reciprocal table [OL 91], or applying bipartition [DM 95] or interpolation [DM 96, Fa 81, IT 95, Na 87, WG 95] with a moderate sized such table, provides an easy to implement efficient alternative to a division instruction. In particular, a single precision floating point reciprocal instruction as a short latency operation with a moderately sized hardware implementation is very attractive for fast 3-D graphics. For high performance implementation of IEEE standard 754 double or double extended floating point division, the faster multiplier based iterative division algorithms such as Newton Raphson, convergence, prescaled, and short reciprocal method [BM 93, BM 95, DG 89, FS 89, Ka 91] initially employ a a seed reciprocal of the divisor typically provided by a reciprocal table.

The limitation of the conventional reciprocal tables is that an attempt to enhance the accuracy of the seed reciprocal by just one bit results in more than doubling the table size. How can one obtain more accurate reciprocals at acceptable costs in time and area? This question represents a research area with important implications in the designs of fast division algorithms for graphics and floating point units. Our contribution to this end is to define and show how to construct *finite precision linear interpolation in reciprocal tables.*

The accuracy standard for our interpolated reciprocal table output is obtained in that the reciprocal provided will be guaranteed to be correct to a unit in the last place, *i.e.* less than one ulp deviation from the infinitely precise reciprocal of the arbitrarily precise input argument, As a secondary goal, we also attempt to maximize the percentage of input arguments that are rounded to nearest. To provide standards for measuring compression and overall accuracy, we first provide the optimal results in table size minimization and portion round to nearest maximization that can be obtained by conventional reciprocal tables.

Reciprocal tables are generally constructed by assuming that the argument is normalized $1 \leq x < 2$ and truncated to k bits to the right of the radix point, $trunc(x) = 1.b_1b_2...b_k$. These k bits are used to index a table providing m output bits which are taken as the m bits after the leading bit in the m+1 bit fraction reciprocal approximation $recip(x) = 0.1b'_1b'_2...b'_m$. Such a table is termed a k-bits-in m-bits-out reciprocal table of size $2^k m$ bits. The accuracy of reciprocal tables was thoroughly investigated by us in [Da 95] and [DM 94].

In [DM 95] we defined as faithful those i-bits-in j-

1063-6889/97 $10.00 © 1997 IEEE

bits-out reciprocal tables for which the output always satisfies a one ulp bound, and as max RN those tables which maximize the portion of input $1 \leq x < 2$ which yields an output reciprocal that is a round-to-nearest value of $\frac{1}{x}$. In [DM 95] we also presented the *midpoint reciprocal algorithm* which generates minimum sized tables to guarantee faithful reciprocals for each table entry, and for faithful tables maximizes the percentage of input values obtaining round-to-nearest output. We showed that for $g \geq 1$ indicating a number of input guard bits, the $j + g$-bits-in, j-bits-out reciprocal tables generated by the midpoint reciprocal algorithm are both faithful and max RN for any $g \geq 1$ and all $j \geq 1$. The percentage round-to-nearest was shown to be of the order 88% even with the smallest number $g = 1$ of input guard bits for which one ulp guaranteed output is obtained.

In Section 2 we first define finite precision linear interpolation and the errors associated with it. We then describe our finite precision interpolation method that employs a compressed reciprocal look-up table and a small sized multiplier to yield an output reciprocal as a simple direct operation.

A $(2k + g)$-bits-in, $2k$-bits-out interpolated reciprocal is constructed by assuming the infinitely precise argument is normalized $1 \leq x < 2$ and truncated to $2k + g$ places to the right of the radix point, $trunc(x) = 1.b_1b_2...b_{2k+g}$. The interpolation procedure is described by reference to the logic circuitry illustrated in Figure 1. The figure shows the $2k$+3-bit input $trunc(x)$ and a $2k$-bit faithful reciprocal $recip(x) = 0.1b_1'b_2'...b_{2k}'$ as output. The interpolation steps are:

1. the $2k + 3$ bits of the truncated input argument $trunc(x)$ are partitioned into high, and low fields, x_h and f, of sizes k and $k + 3$ bits respectively,

2. the leading (high) k input bits x_h index a reciprocal table which yields a $2k + 2$-bit table output $c_1(x_h)$ and implicitly the difference of successive table outputs $c_2(x_h) = c_1(x_h) - c_1(x_h + \frac{1}{2^k})$ in the borrow-save form which are fed into the Multiply/Add unit as addend and multiplier inputs respectively,

3. the $k + 3$ fractional (low) bits f are fed into the Multiply/Add unit as the multiplicand input,

4. The interpolated reciprocal of the $2k + 3$-bit input $trunc(x) = x_h + 2^{-k} \times f$ is computed by the fused product sum operation $recip(x) = c_1(x_h) - c_2(x_h) \times f$ in the Multiply/Add unit. The low order guard bits of the product sum are rounded off (chopped) to obtain a $2k$-bit reciprocal value guaranteed correct to 1 ulp.

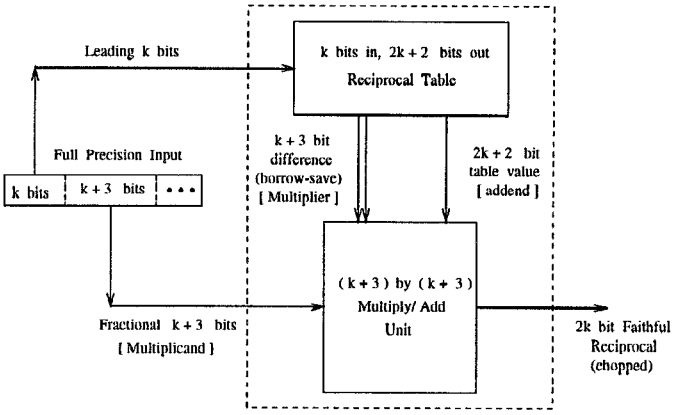

Figure 1. A 2k+3-bits-in 2k-bits-out faithful interpolated reciprocal

Our principal result in Section 3 is that our interpolation method described in Section 2 requires only a few input and table guard bits to guard against the finite precision errors compounding with the theoretical real interpolation error and failing to guarantee faithful reciprocals. Specifically, the interpolation method is proved to generate $2k$-bit faithful reciprocals employing a k-bits-in $2k$+2-bits-out table and a $(k+3)\times(k+3)$ bit multiplier. A single precision faithful reciprocal can thus be generated employing a table of size 13 Kbytes and a 15 × 15 bit multiplier, compared to table sizes of 46 Mbytes and 544 Kbytes for conventional reciprocal tables and bipartite tables [DM 95] respectively. The use of such a small sized multiplier allows the interpolation process to yield a single precision faithful reciprocal with only a couple of cycle latency.

In Section 4 we describe a method which modifies the table construction method presented in Section 2 to improve the percentage of inputs which obtain *round-to-nearest* value of the interpolated reciprocal still preserving the 1 ulp accuracy bound. Our principal result in Section 4 is that some 92% of the $2k$-bit interpolated reciprocals obtain round-to-nearest value compared to only about 80% with the unmodified algorithm of Section 2 and an upper bound of some 97% of the $2k$-bit reciprocals from an optimal $2k + 3$-bits-in $2k$-bits-out reciprocal table. This is obtained despite the great variance of the sizes of the tables employed.

In Section 5 we present some concluding remarks and compare our table compression and accuracy with some other methods of interpolation in the literature. We also briefly discuss the possible application of our interpolation method for computing values of other mathematical functions.

2. Finite Precision Linear Interpolation

Linear interpolation is a well-known technique to approximate a non-linear function such as the reciprocal function [DM 96, Fa 81, Fe 67, IT 95, Na 87, SO 93] by a linear function. Consider $f(x) = \frac{1}{x}$ to be the reciprocal function in the interval $[a, b]$. Let $g(x) = c_1 + c_2 x$ be the linear approximation to $f(x)$ in $[a, b]$. The process of approximating $f(x)$ by $g(x)$ is called linear interpolation. More precisely when x, c_1, c_2, and $g(x)$ are all reals (infinitely precise), we refer to $g(x)$ as the infinite precision interpolation of $f(x)$ in $[a, b]$. Figure 2 shows such an interpolation for reals. The dashed line C represents the interpolated values of the reciprocals. The only error involved in such an infinite precision interpolation for an input x is the difference $g(x) - f(x)$ which we call interpolation error. A closed form solution to determine the optimal values of c_1 and c_2 which minimize the maximum relative error in the interval $[a, b]$ is given in [SO 93]. The minimum value of the maximum absolute error for the infinite precision interpolation can be easily found with reference to Figure 2. Line C in Figure 2 minimizes the maximum absolute error of interpolation in the interval $[a, b]$. Line C is constructed parallel to the line A formed by joining the points a and b and line B formed by drawing a tangent to the reciprocal curve with the same slope, which occurs at the geometric mean, $\sqrt{ab}$. Line C is equidistant from line A and line B. The computation of the optimal values

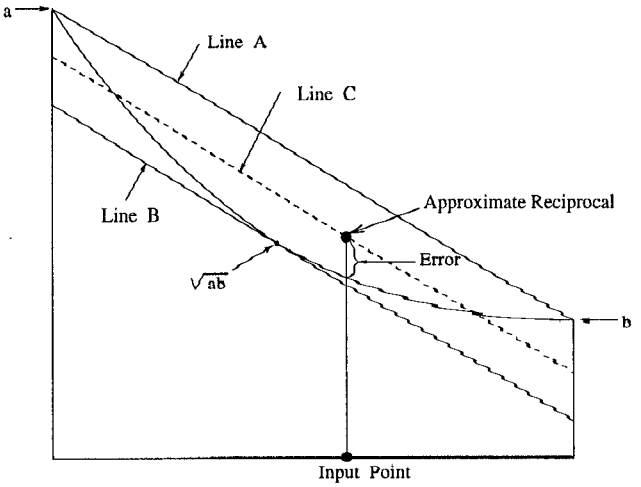

Figure 2. Infinite precision interpolation of reciprocals

of c_1, and c_2 for interpolation in reciprocal tables to compute a reciprocal of finite length is more involved and the optimal values of c_1 and c_2 for infinite precision interpolation [Fe 67, SO 93] is not applicable to the finite case where only finite values of x, c_1, and c_2 can be stored. Three additional errors are incurred in finite precision interpolation: the first due to the rounding of the table values, the second due to rounding of the interpolated value, and the third because of the fact that each interpolated value must represent the reciprocal for an input interval rather than an input point. The errors involved in such a finite precision interpolation in reciprocal tables are shown in Figure 3.

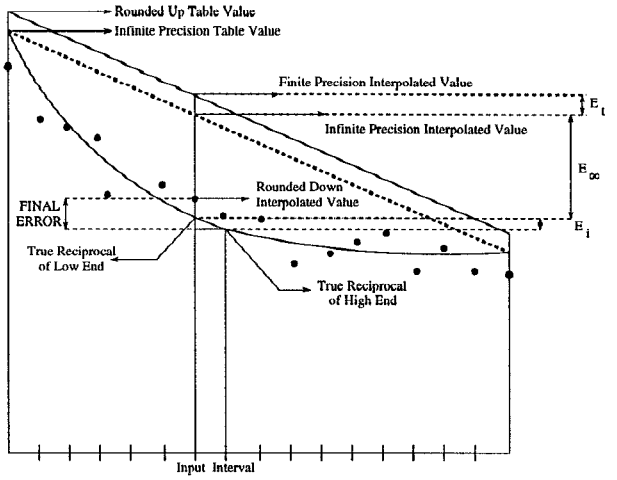

Figure 3. Finite precision interpolation in reciprocal tables

Figure 3 shows one piece for the piecewise linear interpolation implicit for interpolation in a reciprocal table. The dashed straight line joining the end points of the piece of the reciprocal curve shown represents the best upper bound infinite precision interpolation utilized which is guaranteed to upperbound the true result. At any given point, the vertical distance between the true reciprocal curve and the dashed straight line is the error due to best upper bound interpolation and is called *interpolation error* E_∞. However, the table values at the two ends of the piece are finite in length and may be subjected to rounding. The solid straight line shows the finite precision upperbound interpolation in the rounded up table values. The vertical distance between the dashed line and the solid line at any given point in the figure illustrates the independent additional rounding error due to the finiteness of the table values and is called *table discretization error* E_t. Also the true reciprocal at any given point on the reciprocal curve really represents the reciprocal for the finite input interval rather than just that input point because of the truncated input used to compute the re-

ciprocal. The vertical distance between the reciprocal curve values at the end points of the input interval is called the *input discretization error* E_i and depends on the width and curvature of the input interval. Finally the interpolated reciprocals on the solid line are each subjected to rounding (down) to yield finite precision reciprocals shown in the figure as solid dots. For any input interval, the maximum vertical distance between the finite precision rounded interpolated reciprocal for that interval and a reciprocal of any point in the input interval is called the *total error*. Our objective in this paper is to construct reciprocal tables (c_1 and implicit c_2) such that each finite precision interpolated reciprocal differs from the true reciprocal by less than a unit in the last place. Henceforth, unless otherwise mentioned, interpolation refers to this compounded error finite precision interpolation process.

We now describe the construction of a k-bits-in $2k+g_t$-bits-out reciprocal table with table output c_1 ($2k+g_t$ bits). Given a $2k+g_i$-bit input $trunc(x)$, the $2k$-bit reciprocal of $trunc(x)$ is computed by interpolation in the k-bits-in $2k+g_t$-bits-out reciprocal table where g_i and g_t are the *input guard bits* and *table guard bits* respectively. The interpolation method is formalized by two algorithms. Algorithm 1 formalizes the table output c_1. Algorithm 2 formalizes the computation of the $2k$-bit reciprocal by interpolation.

Algorithm 1 [Table Construction]

Stimulus: Integers $k \geq 1$ and $g_t \geq 1$

Response: A k-bits-in $2k+g_t$-bits-out reciprocal table with $2k+g_t$-bit output c_1

Method: **for** $i = 2^k$ to 2^{k+1} **step** 1 <for each input interval $[\frac{i}{2^k}, \frac{i+1}{2^k})$>

begin

L1: $c_1(i) := RU(\frac{2^k}{i} 2^{2k+g_t+1})$ <round-up the reciprocal of the low end of the interval; rounding is to an integer with assumed division by 2^{2k+g_t+1}; the integer is always a $2k+g_t+1$ bit string of which the leading bit need not be stored>

end

Algorithm 2 [Reciprocal by Interpolation in Table]

Stimulus: $2k+g_i$-bit input $trunc(x)$, k-bits-in $2k+g_t$-bits-out reciprocal table,

Response: A $2k$-bit interpolated reciprocal $recip(x)$

Method: **begin**

L1: $trunc(x) = x_h + 2^{-k} f$ where $trunc(x) = \frac{j}{2^{2k+g_i}}$, $x_h = \frac{i}{2^k}$, and $f = \frac{l}{2^{k+g_i}}$ <partition $trunc(x)$ into k-bit leading part x_h, and $k+g_i$-bit fractional part f>

L2: $recip(y) = RZ(c_1(i) - (c_1(i) - c_1(i+1)) \times f)$ to $2k$ bits <lookup the table outputs for x constructed by Algorithm 1, interpolate in the table proportional to the fractional part f, and round-to-zero (chop) the product sum to $2k$ bits>

end

Note that the difference $c_1(i) - c_1(i+1)$ is implicitly defined in the table by the pair of successive table outputs $[c_1(i), c_1(i+1)]$ in borrow-save form and need not be stored separately in the table. Thus the size of the table is only $2^k \times (2k+g_t)$ bits. The time to access two successive table outputs, and to recode the borrow-save value in the multiplier increases the execution time only marginally [LM 95]. Nakano [Na 87] and Ito *et al* [IT 95] use two tables to store the table outputs and the differences between successive pair of table outputs in his interpolation method resulting in unnecessarily larger total table size. Also as the following lemma shows, the leading k bits of $c_1(i) - c_1(i+1)$ for $2^k \leq i \leq 2^{k+1} - 1$ are zeros. This implies that with proper encoding of the extracted borrow-save value [DM 97], a $(k+g_t+1) \times (k+g_i)$ multiplier is sufficient for the interpolation.

Lemma 1 *Let* $c_1(i) = RU(\frac{2^k}{i})$ *to* $2k+g_t$ *bits and* $c_2(i) = c_1(i) - c_1(i+1)$ *be the difference of two successive table entries where* $2^k \leq i \leq 2^{k+1} - 1$. *Then the leading* k *fractional bits of* $c_2(i)$ *are zeros.*

Proof: The difference between the infinite precision values of $c_1(i)$ and $c_1(i+1)$ is $\frac{2^k}{i} - \frac{2^k}{i+1}$. But since the table entries $c_1(i)$ and $c_1(i+1)$ are rounded up to $2k+g_t$ bits, $c_2(i) \leq \frac{2^k}{i} - \frac{2^k}{i+1} + \frac{1}{2^{2k+g_t+1}} \leq \frac{2^k}{i(i+1)} + \frac{1}{2^{2k+g_t+1}} \leq \frac{2^k}{2^k(2^k+1)} + \frac{1}{2^{2k+g_t+1}} \leq \frac{1}{2^k} - \frac{1}{2^k(2^k+1)} + \frac{1}{2^{2k+g_t+1}} < \frac{1}{2^k}$, since $g_t \geq 0$. Thus the leading k fractional bits of $c_2(i)$ are zeros proving the claim. □

We apply Algorithm 1 to construct a 2-bits-in 6-bits-out table with $k=2$, and $g_t=2$ and illustrate the results in Table 1. Consider for example line two of Table 1. Line two is indexed by the two bit string 01 arising from input truncated to the value 1.01. This truncated input 1.01 represents the arbitrarily precise input arguments falling in the input interval $[1.01, 1.10)$. The reciprocal of the low end 1.01 of this interval is computed as 0.1100110 011... and rounded-up to 6 bits to yield the value 0.1100111 for c_1.

Table 1. 2-bits-in 6-bits-out reciprocal table constructed by Algorithm 1

input x	$c_1(x)$
1.00 xxxxx	1.0000000
1.01 xxxxx	0.1100111
1.10 xxxxx	0.1010110
1.11 xxxxx	0.1001010

We then apply Algorithm 2 to generate 4-bit reciprocals of 7-bit input operands (with $g_i = 3$) by piecewise interpolation in the 2-bits-in 6-bits-out table constructed in Table 1. There are four pieces of interpolation corresponding to the four table entries in Table 1. Between two successive table entries in Table 1, $2^5 = 32$ values are interpolated generating reciprocals of 32 7-bit input arguments contained in each input interval of Table 1. Consider for example a 7-bit input operand 1.0000111. The first 2 fraction bits of the input operand 1.0000111, :00... is used as an index to the (2,6) bit Table 1, whose first table output is $c_1 = 1.0000000$. The difference of the table outputs for indices 1.00 and 1.01 is 1.0000000 - 0.1100111 = 0.0011001 which we call c_2. Note that the leading two bits of c_2 are zeros as proved in Lemma 1. The last 5 fractional bits $f = 0.00111$ of the input operand 1.0000111 is multiplied with c_2 to get the term $c_2 \times f$ which is then subtracted from c_1 with the fused product sum operation and then rounded-down to 4 bits to yield the value 0.11110 as a reciprocal of the input operand 1.0000111. Note that a 2-bits-in, 6-bits-out reciprocal table and a 5 × 5 multiplier are required for this interpolation. Each of the 4-bit reciprocals of the 7-bit input arguments generated by Algorithm 1 and Algorithm 2 is faithful. In the following section we show that the $2k$-bit reciprocals of $2k + 3$-bit input arguments obtained by interpolation in the k-bits-in $2k + 2$-bits-out table constructed by Algorithm 1 are guaranteed to be faithful.

3. Faithful Reciprocal by Interpolation

In this section we show that only a few input and table guard bits are required to guard against the input discretization error and table discretization error compounding with the theoretical real interpolation error and failing to guarantee faithful reciprocals.

We first formally define the three errors, namely interpolation error, table discretization error, and input discretization error associated with our finite precision linear interpolation, which we explained earlier with reference to Figure 3.

Consider any $2k + g_i$-bit input argument $\frac{j}{2^{2k+g_i}}$ where $2^{2k+g_i} \leq j < 2^{2k+g_i+1}$ whose $2k$-bit reciprocal is generated by the interpolation described in Algorithm 2 in a k-bits-in $2k + g_t$-bits-out table constructed by Algorithm 1. Let T_j be the infinitely precise reciprocal of $\frac{j}{2^{2k+g_i}}$, and I_j be the interpolated value of its reciprocal assuming the table outputs to be infinitely precise, and $I_j^{(g_t)}$ be the interpolated value of the reciprocal where the table outputs are rounded up to $2k + g_t$ bits.

$E_\infty^{(j)} = I_j - T_j$ in ulps is defined as the *interpolation error*. $E_\infty^{(j)} \geq 0$ since the reciprocal function $f(x)$ is a convex function with the slope monotonically decreasing in $1 \leq x < 2$. $E_\infty = max_j\ E_\infty^{(j)}$ is defined as the *maximum interpolation error.*

$E_t^{(j)} = I_j^{(g_t)} - I_j$ in ulps is defined as the *table discretization error.* $E_t^{(j)} \geq 0$ since the table outputs are rounded up in Algorithm 1. $E_t = max_j\ E_t^{(j)}$ is defined as the *maximum table discretization error.*

Finally $E_i^{(j)} = T_j - T_{j+1}$, the difference in ulps between the reciprocals of the two ends of the input interval $[\frac{j}{2^{2k+g_i}}, \frac{j+1}{2^{2k+g_i}})$ is defined as the *input discretization error.* $E_i^{(j)} \geq 0$ since T_j decreases monotonically with increase in j. $E_i = max_j\ E_i^{(j)}$ is defined as the *maximum input discretization error.*

$ERROR_j = I_j^{(g_t)} - T_{j+1} = E_\infty^{(j)} + E_t^{(j)} + E_i^{(j)}$ is the total error in ulps in computing the unrounded reciprocal of the j^{th} input argument $\frac{j}{2^{2k+g_i}}$ by our method of linear interpolation. Clearly $ERROR_j \geq 0$. Also, if $ERROR_j < 1$ for all j, then the rounded down value of the interpolated reciprocal generated by Algorithm 2 will be faithful since the rounding error introduced in the final round-down operation is less than 1 ulp.

Table 2 shows the values of T_j, I_j, and $I_j^{(g_t)}$, and Table 3 shows the values of $E_\infty^{(j)}$, $E_t^{(j)}$, $E_i^{(j)}$, and $ERROR_j$ for the first piece of interpolation in a 2-bits-in, 6-bits-out table to produce 4-bit faithful reciprocals of 7-bit input operands. Consider for example line eight of Table 2. It corresponds to the 7-bit chopped input $\frac{135}{128}$. The infinite precision reciprocal, T_j, the infinite precision interpolated reciprocal, I_j, and the finite precision interpolated reciprocal , $I_j^{(g_t)}$, of this chopped input are given to several decimal places as $\frac{30.3407}{32}$, $\frac{30.6000}{32}$, and $\frac{30.6328}{32}$ respectively. The finite precision interpolated reciprocal is then rounded down to a 4-bit interpolated reciprocal $\frac{30}{32}$. Now consider line eight of Table 3 which corresponds to the interpolation of the chopped input $\frac{135}{128}$ shown in line eight of Table 2. $E_\infty^{(j)}$ given in ulps = 30.6000 - 30.3407 = 0.2593 ulps. $E_t^{(j)}$ given in ulps = 30.6328 - 30.6000 = 0.0328 ulps.

Table 2. First piece of interpolation in 2-bits-in,6-bits-out table to generate 4-bit reciprocals of 7-bit input arguments

Chopped Input ×(1/128)	Inf. Prec. Recip. ×(1/32)	Inf. Interpol. ×(1/32)	Finite Interpol. ×(1/32)	RD Recip. ×(1/32)
128	**32.0000**	**32.0000**	**32.0000**	**32**
129	31.7519	31.8000	31.8047	31
130	31.5077	31.6000	31.6094	31
131	31.2672	31.4000	31.4141	31
132	31.0303	31.2000	31.2188	31
133	30.7970	31.0000	31.0234	31
134	30.5672	30.8000	30.8281	30
135	**30.3407**	**30.6000**	**30.6328**	**30**
136	30.1176	30.4000	30.4375	30
137	29.8978	30.2000	30.2422	30
138	29.6812	30.0000	30.0469	30
139	29.4676	29.8000	29.8516	29
140	29.2571	29.6000	29.6563	29
141	29.0496	29.4000	29.4609	29
142	28.8451	29.2000	29.2656	29
143	28.6434	29.0000	29.0703	29
144	28.4444	28.8000	28.8750	28
145	28.2483	28.6000	28.6797	28
146	28.0548	28.4000	28.4844	28
147	27.8639	28.2000	28.2891	28
148	27.6757	28.0000	28.0938	28
149	27.4899	27.8000	27.8984	27
150	27.3067	27.6000	27.7031	27
151	27.1258	27.4000	27.5078	27
152	26.9474	27.2000	27.3125	27
153	26.7712	27.0000	27.1172	27
154	26.5974	26.8000	26.9219	26
155	26.4258	26.6000	26.7266	26
156	26.2564	26.4000	26.5313	26
157	26.0892	26.2000	26.3359	26
158	25.9241	26.0000	26.1406	26
159	25.7610	25.8000	25.9453	25
160	**25.6000**	**25.6000**	**25.7500**	**25**

Table 3. Errors in first piece of interpolation in 2-bits-in,6-bits-out table to generate 4-bit reciprocals of 7-bit inputs

Chopped Input ×(1/128)	$E_\infty^{(j)}$ (ulps)	$E_t^{(j)}$ (ulps)	$E_i^{(j)}$ (ulps)	$ERROR_j$ (ulps)
128	0.0000	0.0000	0.2481	0.2481
129	0.0481	0.0047	0.2442	0.2970
130	0.0923	0.0094	0.2405	0.3422
131	0.1328	0.0141	0.2369	0.3838
132	0.1697	0.0187	0.2333	0.4218
133	0.2030	0.0234	0.2298	0.4563
134	0.2328	0.0281	0.2264	0.4874
135	**0.2593**	**0.0328**	**0.2231**	**0.5152**
136	0.2824	0.0375	0.2198	0.5397
137	0.3022	0.0422	0.2167	0.5610
138	0.3188	0.0469	0.2135	0.5792
139	0.3324	0.0516	0.2105	0.5944
140	0.3429	0.0562	0.2075	0.6066
141	0.3504	0.0609	0.2046	0.6159
142	0.3549	0.0656	0.2017	0.6223
143	0.3566	0.0703	0.1989	0.6259
144	0.3556	0.0750	0.1962	0.6267
145	0.3517	0.0797	0.1935	0.6249
146	0.3452	0.0844	0.1908	0.6204
147	0.3361	0.0891	0.1883	0.6134
148	0.3243	0.0937	0.1857	0.6038
149	0.3101	0.0984	0.1833	0.5918
150	0.2933	0.1031	0.1808	0.5773
151	0.2742	0.1078	0.1785	0.5604
152	0.2526	0.1125	0.1761	0.5413
153	0.2288	0.1172	0.1738	0.5198
154	0.2026	0.1219	0.1716	0.4961
155	0.1742	0.1266	0.1694	0.4702
156	0.1436	0.1312	0.1672	0.4421
157	0.1108	0.1359	0.1651	0.4119
158	0.0759	0.1406	0.1630	0.3796
159	0.0390	0.1453	0.1610	0.3453
160	0.0000	0.1500	0.1590	0.3090

$E_i^{(j)}$ given in ulps = 30.3407 - 30.1176 = 0.2231 ulps. The total error, $ERROR_j$, for the interpolated reciprocal in line eight is 30.6328 - 30.1176 = 0.5152 ulps which equals $E_\infty^{(j)} + E_t^{(j)} + E_i^{(j)}$. Note that after the interpolated reciprocal is rounded down to 4 bits to yield $\frac{30}{32}$ the maximum error is only 0.3407 ulps making it faithful. For overall reduction of the table and multiplier size, we find minimum numbers of input and table guard bits, g_i and g_t, such that the interpolated reciprocals are faithful. We compute $ERROR$,the theoretical upper bound on $ERROR_j$, as $E_\infty + E_t + E_i$, by considering the upper bounds on the maximum possible values of $E_\infty^{(j)}$, $E_t^{(j)}$, and $E_i^{(j)}$ separately.

Note that these maximum errors can occur for different inputs. However as we will show later, E_∞ and E_i attain their maximum values in the first piece of interpolation, and E_t is statistically random. Also as k increases, the maximum error realized approaches $ERROR$ rapidly, implying $ERROR$ is a tight and a certain true upper bound on the maximum realized error $ERROR_j$ for any j. If $ERROR$ is less than one ulp then the reciprocals are guaranteed to be faithful.

To compute E_∞, we first show that the maximum value of $E_\infty^{(j)}$ in each piece of interpolation is realized for the input argument corresponding to the geometric mean of the end points of the input interval on which the interpolation is performed. Then we show that the

supremum of all these maximum errors occurs at the first piece of interpolation.

Lemma 2 *The maximum interpolation error in any interval occurs for the input argument corresponding to the geometric mean of the end points of the interval.*

Proof: Let x be any input argument in the interval $[a, b)$. Then the interpolated value of the reciprocal of x is given by $g(x) = \frac{1}{a} + \frac{1}{b} - \frac{x}{ab}$, and the true reciprocal of x is given by $f(x) = \frac{1}{x}$. Their difference is given by $h(x) = g(x) - f(x) = \frac{1}{a} + \frac{1}{b} - \frac{x}{ab} - \frac{1}{x}$. $h'(x) = -\frac{1}{ab} + \frac{1}{x^2}$ which is 0 at $x = \sqrt{ab}$. $h''(\sqrt{ab}) = -\frac{2}{(\sqrt{ab})^3}$ which is negative implying $h(x)$ achieves maximum at $x = \sqrt{ab}$ and the maximum value of $h(x)$ is $\frac{1}{a} + \frac{1}{b} - \frac{2}{\sqrt{ab}}$ proving the lemma. □

It is interesting to note that the maximum absolute value of the interpolation error is realized at the *geometric mean* of the end points of an interval while the maximum relative error has been shown to occur at the *arithmetic mean* of the end points of an interval [Fe 67].

Lemma 3 *Let $E_\infty^{(i_m)}$ denote the maximum interpolation error in the i^{th} interval $[1 + \frac{i-1}{2^k}, 1 + \frac{i}{2^k})$ where $1 \leq i \leq 2^k$. Then $E_\infty^{(i_m)}$ monotonically decreases with increase in i and is maximum at $i = 1$.*

Proof: From Lemma 2 the maximum interpolation error in the i^{th} interval $[1 + \frac{i-1}{2^k}, 1 + \frac{i}{2^k})$ occurs at $\sqrt{(1 + \frac{i-1}{2^k})(1 + \frac{i}{2^k})}$, and is obtained as $E_\infty^{(i_m)} = (\frac{1}{\sqrt{1+\frac{i-1}{2^k}}} - \frac{1}{\sqrt{1+\frac{i}{2^k}}})^2 = 2^k(\frac{1}{\sqrt{X}} - \frac{1}{\sqrt{X+1}})^2$ where $X = i - 1 + 2^k$ with $2^k \leq X \leq 2^{k+1} - 1$. $\frac{1}{\sqrt{X}} - \frac{1}{\sqrt{X+1}} = \frac{\sqrt{X+1}-\sqrt{X}}{\sqrt{X(X+1)}} = \frac{1}{\sqrt{X(X+1)}(\sqrt{X+1}+\sqrt{X})}$ which clearly decreases monotonically with increase in X and is maximum at $X = 2^k$. Thus $E_\infty^{(i_m)}$ is maximum at $i = 1$, proving the lemma. □

Thus from Lemma 2 and Lemma 3, the maximum value of $E_\infty^{(j)}$ for any input operand $1 + \frac{j}{2^{2k+g_i}}$, where $0 \leq j \leq 2^{2k+g_i} - 1$ is realized at the geometric mean of the end points of the input interval $[1, 1 + \frac{1}{2^k})$ corresponding to the first piece of interpolation. In the next lemma we derive the upper bound E_∞ on $E_\infty^{(j)}$.

Lemma 4 *The maximum value of $E_\infty^{(j)}$ for any input operand $1 + \frac{j}{2^{2k+g_i}}$, where $0 \leq j \leq 2^{2k+g_i} - 1$ is upper bounded by $E_\infty = \frac{1}{2} - \frac{3}{2^{k+2}} + \frac{1}{2^{2k}}$ in ulps.*

Proof: From Lemma 2 and Lemma 3, the maximum value of $E_\infty^{(j)}$ for any input operand $1 + \frac{j}{2^{2k+g_i}}$, where $0 \leq j \leq 2^{2k+g_i} - 1$, is realized at the geometric mean of the end points of the input interval $[1, 1 + \frac{1}{2^k})$ corresponding to the first piece of interpolation and is given in ulps as $(1 + \frac{2^k}{2^{k+1}} - \frac{2}{(1+\frac{1}{2^k})^{\frac{1}{2}}})2^{2k+1} = (1 + (1 + \frac{1}{2^k})^{-1} - 2(1 + \frac{1}{2^k})^{-\frac{1}{2}})2^{2k+1} = (1 + (1 - \frac{1}{2^k} + \frac{1}{2^{2k}} - \frac{1}{2^{3k}} + \frac{1}{2^{4k}} - ...) - 2(1 - \frac{1}{2^{k+1}} + \frac{3}{2^{2k+3}} - \frac{5}{2^{3k+4}} + \frac{35}{2^{4k+7}} - ...))2^{2k+1} = ((\frac{1}{2^{2k}} - \frac{3}{2^{2k+2}}) - (\frac{1}{2^{3k}} - \frac{5}{2^{3k+3}}) + (\frac{1}{2^{4k}} - \frac{35}{2^{4k+6}}) - ...)2^{2k+1} < (\frac{1}{2^{2k+2}} - \frac{3}{2^{3k+3}} + \frac{29}{2^{4k+6}})2^{2k+1} < \frac{1}{2} - \frac{3}{2^{k+2}} + \frac{1}{2^{2k}}$ Thus $E_\infty = \frac{1}{2} - \frac{3}{2^{k+2}} + \frac{1}{2^{2k}}$ in ulps, proving the lemma. □

In the next lemma we derive E_t, the upper bound on $E_t^{(j)}$.

Lemma 5 *The maximum value of $E_t^{(j)}$ for any input operand $1 + \frac{j}{2^{2k+g_i}}$, where $0 \leq j \leq 2^{2k+g_i} - 1$ is upper bounded by $E_t = \frac{1}{2^{g_t}}$ in ulps.*

Proof: The rounding error for any interpolated reciprocal is at most $\frac{1}{2^{g_t}}$ ulps since each table entry is rounded up to $2k + g_t$ bits. So trivially $E_t = \frac{1}{2^{g_t}}$ in ulps, proving the lemma. □

In the following lemma, we compute E_i by first showing that $max_j E_i^{(j)}$ is realized for the first input argument in the table, and then upper bounding that value by E_i.

Lemma 6 *The maximum value of $E_i^{(j)}$ for any input operand $1 + \frac{j}{2^{2k+g_i}}$, where $0 \leq j \leq 2^{2k+g_i} - 1$ is realized at $j = 0$ and is upper bounded by $E_i = \frac{1}{2^{g_i-1}}$ in ulps.*

Proof: $E_i^{(j)}$ is the difference in ulps between the reciprocals of the end points of the input interval $[1 + \frac{j}{2^{2k+g_i}}, 1 + \frac{j+1}{2^{2k+g_i}})$, where $0 \leq j \leq 2^{2k+g_i} - 1$ and is given by $E_i^{(j)} = \frac{2^{2k+g_i}}{2^{2k+g_i}+j}2^{2k+1} - \frac{2^{2k+g_i}}{2^{2k+g_i}+j+1}2^{2k+1} = \frac{2^{2k+g_i}2^{2k+1}}{(2^{2k+g_i}+j)(2^{2k+g_i}+j+1)}$ which is clearly maximum at $j = 0$. Thus $max_j E_i^{(j)} = (1 - \frac{2^{2k+g_i}}{2^{2k+g_i}+1})2^{2k+1} = (1 - (1 + \frac{1}{2^{2k+g_i}})^{-1})2^{2k+1} = (1 - 1 + \frac{1}{2^{2k+g_i}} - \frac{1}{2^{4k+2g_i}}(1 + \frac{1}{2^{2k+g_i}})^{-1})2^{2k+1} = \frac{1}{2^{g_i-1}} - \frac{1}{2^{g_i-1}(2^{2k+g_i}+1)} < \frac{1}{2^{g_i-1}}$ Thus $E_i = \frac{1}{2^{g_i-1}}$ in ulps, proving the lemma. □

In the next lemma we derive the upper bound $ERROR$ on the maximum total error $ERROR_j$.

Lemma 7 *The maximum value of $ERROR_j$ is upper bounded by $ERROR = (\frac{1}{2} - \frac{3}{2^{k+2}} + \frac{1}{2^{2k}}) + \frac{1}{2^{g_t}} + \frac{1}{2^{g_i-1}}$ in ulps.*

Proof: $ERROR = E_\infty + E_t + E_i$, where E_1, E_t, and E_i are as derived in Lemma 4, Lemma 5, and Lemma 6 respectively. Substitution yields $ERROR = (\frac{1}{2} - \frac{3}{2^{k+2}} + \frac{1}{2^{2k}}) + \frac{1}{2^{g_t}} + \frac{1}{2^{g_i-1}}$ ulps, proving the lemma. □

In the following theorem we find the suitable values of g_i, and g_t such that interpolated reciprocals are faithful for any input argument.

Theorem 1 *A $2k$-bit interpolated reciprocal of any $2k+3$-bit input operand by interpolation in a k-bits-in, $2k+2$-bits-out table is faithful.*

Proof: It is immediate from Lemma 7 that any $2k$-bit interpolated reciprocal is faithful if $ERROR \leq 1$ ulp. Clearly if $g_i = 3$, and $g_t = 2$, then $ERROR \leq 1$, and so the rounded down value of the interpolated reciprocal will indeed be faithful, proving the theorem. □

Theorem 1 also suggests that k bits is indeed the minimum number of bits needed to index the table to guarantee all the $2k$-bit interpolated reciprocals to be faithful irrespective of the number of guard bits g_i, and g_t employed. If only $k-1$ bits are used to index the table, then following an analysis similar to Lemma 4 we obtain the maximum interpolation error to be greater than $2 - \frac{6}{2^k}$ ulps which is greater than 1 ulp for any $k > 2$. So $ERROR$ will exceed 1 ulp proving the following corollary.

Corollary 1 *A $2k$-bit interpolated reciprocal of any $2k+g_i$-bit input operand by interpolation in a $k-1$-bits-in, $2k+g_t$-bits-out table is not guaranteed to be faithful for any $g_i \geq 0$, and $g_t \geq 0$ where $k > 2$.*

Observe that to generate a $2k$-bit faithful reciprocal by interpolation, we need a k-bits-in, $2k+2$-bits-out table and a $(k+3) \times (k+3)$ bit multiplier. The k-bits-in, $2k+2$-bits-out reciprocal table requires $2^k(2k+2)$ bits of storage. Compare this with the size of $2k+1$-bits-in, $2k$-bits-out reciprocal table to generate $2k$-bit faithful reciprocals directly form a table, which requires $2^{2k+1}(2k)$ bits of storage. Thus generating a single precision faithful reciprocal by interpolation in the compressed table requires only a 13 Kbytes table compared to a 46 Mbytes table for direct lookup, yielding a compression factor more than 3600:1. Also the small sized multiplier (15×15 bit) allows the table look-up followed by interpolation to have only a couple of cycle latency.

4. Optimizing Percentage of Inputs RN

In the last section, we showed that $2k$-bit interpolated reciprocals of $2k+3$-bit inputs are guaranteed to be faithful. In this section we show that the percentage of inputs which obtain round-to-nearest (RN) value of the reciprocal is over 90%, comparable to that of the large optimal tables. The percentage of inputs round-to-nearest for interpolated reciprocals of length 8 to 24 bits is around 80% from the straightforward application of the Algorithm 1 and Algorithm 2. The relatively poor performance of the percentage RN is because the table outputs were constructed by Algorithm 1 only to guarantee the 1 ulp bound for the interpolated reciprocals. No measure was taken to optimize the table outputs such that the interpolated reciprocals match the corresponding optimum reciprocals constructed by midpoint reciprocal table [DM 94]. The theoretical error bound of 1 ulp for the pre-rounded interpolated reciprocals is realized in actuality only for the early pieces of interpolation where the input operands are close to unity. Thus for those inputs the rounded down reciprocals are close to the corresponding optimum reciprocals yielding high percentage of inputs rounded to nearest. But as the reciprocal curve becomes flatter , the actual errors realized drop much below the theoretical bound of 1 ulp, and the pre-rounded interpolated reciprocals approach the corresponding infinite precision midpoint reciprocals. Consequently for those input arguments, the final rounded down interpolated reciprocals are likely to be 0.5 to 1 ulp below the corresponding infinite precision midpoint reciprocals, resulting in a relatively large percentage of those inputs not rounded-to-nearest compared to the reciprocals obtained from large optimal tables.

To obtain an improved percentage of inputs round to nearest in our interpolation, the pre-rounded interpolated reciprocal values should be above the corresponding infintely precise middle point reciprocals by 0.5 ulp on an average so that the rounded down interpolated reciprocals match the round-to-nearest values of the midpoint reciprocals with high probability. In the modified algorithm, to improve the percentage of inputs RN, we add a compensation factor to each table entry such that between each pair of such table entries the sum of the average interpolation error and the average input discretization error is very close to 0.5 ulp. The average interpolation error between any table entry is about two-thirds the value of the corresponding maximum interpolation error, and the average input discretization error between any table entry is about half the value of the corresponding maximum input discretization error. The compensation factor for each table entry is computed from the errors in its two adjacent input intervals and is derived in [Da 95]. Herein, we just describe the algorithm for computing the modified table entries with the compensation factors.

Algorithm 3 [Table Construction with Compensation Factors]

Stimulus: Integer $k \geq 1$

Response: A k-bits-in $2k+g_t$-bits-out table with the compensation factors added to each table entry as and if required.

Method: L0: $table_0 := 2^{2k+g_t+1}$ <No compensation

factor for the first table entry>

for $i = 1$ to 2^k **step** 1 <for each input interval $[\frac{i}{2^k}, \frac{i+1}{2^k})$>

begin

L1: $a := 1 + \frac{i-1}{2^k}$; $b := 1 + \frac{i}{2^k}$; $c := 1 + \frac{i+1}{2^k}$ <Consider two successive input intervals $[a, b)$ and $[b, c)$ >

L2: $error_i^{(1)} := (\frac{b^2+3ac}{4abc} + \frac{1}{b-a} log_e(\frac{a+b}{b+c}))2^{2k+1}$ <average interpolation error>

L3: $error_i^{(3)} := \frac{2^{2k}}{b.2^{2k+g_i+1}}$ <average input discretization error>

L4: $correct_i$ = 0.5 - $error_i^{(1)}$ - $error_i^{(3)}$ <compensation factor for table entry $table_i$>

L5: $ERROR_i = \frac{1}{2b^3} + \frac{1}{2^{g_i-1}b^2} + \frac{1}{2^{g_t}}$ <original error bound for each interval>

if $ERROR_i + correct_i > 1$ ulp

L6: **then** $table_i := RU((\frac{1}{b}2^{2k+1}) \times 2^{g_t})$ <table entry not modified>

L7: **else** $table_i := RU((\frac{1}{b}2^{2k+1} + correct_i) \times 2^{g_t})$ <modified table entry>

end

Interpolation is then performed in the k-bits-in, $2k+g_t$-bits-out table with the modified table entries as described in Algorithm 3 to generate $2k$-bit faithful reciprocals of $2k + g_i$-bit input arguments. The percentage of inputs not round to nearest for the interpolated reciprocals on the table constructed by Algorithm 3 are computed for some values of k, g_i, and g_t and are shown in Table 4. Note that the percentages round to nearest

Table 4. Percentage of inputs not RN

$2k$	g_i=3,g_t=2	g_i=4,g_t=2	g_i=3,g_t=3
4	8.419	8.438	5.374
6	8.896	6.934	6.622
8	7.405	6.769	5.752
10	7.851	6.828	6.280
12	7.772	7.161	6.120
14	7.367	6.673	6.157
16	7.535	7.119	6.078

in Table 4 compare favorably with that of the optimal $2k + 3$-bits-in,$2k$-bits-out tables [DM 94]. Particularly the percentage not round to nearest of the $2k$-bit reciprocals of $2k + 3$-bit input arguments by interpolation in k-bits-in, $2k + 3$-bits-out table as constructed by Algorithm 3 is always within a factor of two of the percentage not round to nearest of the optimal $2k + 3$-bits-in,$2k$-bits-out table despite the great variance in size of the table utilized. The former requires only $2^k(2k+3)$ bits of storage whereas the latter requires as much as $2^{2k+3}(2k)$ bits of storage. Thus interpolation in the modified table not only guarantees faithful reciprocals but also ensures that the percentage of inputs that are round to nearest is very close to optimal.

5. Conclusion

We described an interpolation method that employs a compressed look-up table and a small sized multiplier to yield an output reciprocal as a simple direct operation. The interpolation method generates single precision faithful reciprocal employing a table of size 13 KBytes and a 15 × 15 multiplier. In [Da 95] we developed a method to compress the table further with a simple scheme based on partitioning the table for different ranges of the input operand. Interpolation on such a partitioned table yields single precision faithful reciprocal employing a total table size of 7.5 KBytes and a 17 × 15 multiplier.

There are other methods of interpolation used in the literature. It is difficult to do a fair comparison with those methods since none of them demonstrate faithfulness of the interpolated reciprocal assuming arbitrarily precise real inputs. We are still able to demonstrate more reduction in table size compared to their methods. Farmwald [Fa 81] and Wong *et al* [WG 95] use many look-up tables to form the approximation. Their methods are based on expanding a Taylor series and using several terms whose coefficients are stored in the tables. In [Fa 81], the tables in conjunction with parallel multipliers are used to speed up the convergence of the Taylor series. In [WG 95], the table values are used in conjunction with Wallace tree of multioperand adders (but no multipliers) to perform Add-Table Lookup-Add (ATA) to yield single precision function values. However both these methods result in larger sized tables. In the ATA method [WG 95] the total table size to yield single precision reciprocal is 8,68,352 bits compared our table size of only 1,06,496 bits. The table size in [Fa 81] is even larger than that of the ATA method [WG 95]. Nakano [Na 87] and Ito *et al* [IT 95] proposed linear interpolation schemes with both the approximate reciprocals of the divisors and the differences between successive approximate reciprocal divisors having to be stored in the table. This results in a larger sized table compared to our interpolation method since we only store the reciprocal divisors. The differences between successive reciprocal divisors are implicitly given in the

borrow-save form.

Our compression method exploits the mathematical properties of the reciprocal function and are independent of the logic minimization which can be used subsequently to further reduce the size of the equivalent PLA. Also our interpolation method can be effectively used to compute values for other special functions such as square root, sine, cosine, tangent, arctangent, logarithm and exponential, and is currently under investigation. Preserving monotonicity in the approximations of these functions is very important along with the 1 ulp accuracy for 3-D graphics library. We have developed some techniques for such monotone interpolations and this topic is currently under further investigation. We believe that a single precision library of faithful and monotone interpolation of special functions implemented in hardware with compressed tables and as a short latency operation is very attractive for 3-D graphics and augers well for its inclusion in the floating point unit designs.

References

[BM 93] W. B. Briggs and D. W. Matula, "A 17×69 Bit Multiply and Add Unit with Redundant Binary Feedback and Single Cycle Latency," in *Proc. 11th IEEE Symp. Comput. Arithmetic*, 1993, pp 163-170.

[BM 95] W. B. Briggs and D. W. Matula, "Method and Apparatus for Prescaled Division," in *United States Patent*, No. 5,475,630, 1995.

[Da 95] D. Das Sarma, "Highly Accurate Initial Reciprocal Approximation for High Performance Division Algorithms," in *Ph. D. thesis, Southern Methodist University*, 1995.

[DG 89] H. M. Darley, M. C. Gill et al, "Floating Point/Integer Processor With Divide and Square Root Functions," in *United States Patent*, No. 4,878,190, 1989.

[DM 94] D. Das Sarma and D. W. Matula, "Measuring the Accuracy of ROM Reciprocal Tables," in *IEEE Trans. Comput.* , Vol. 43, No. 8, August 1994, pp 932-940. Also see *Proc. 11th IEEE Symp. Comput. Arithmetic*, 1993, pp 95-102.

[DM 95] D. Das Sarma and D. W. Matula, "Faithful Bipartite ROM Reciprocal Tables," in *Proc. 12th IEEE Symp. Comput. Arithmetic*, 1995, pp 17-28.

[DM 96] D. Das Sarma and D. W. Matula, "Hardware Reciprocal Table Compression/Decompression Techniques," in *Scientific Computing and Validated Numerics*, pp. 11-17, Akademic Verlag, 1995.

[DM 97] M. Daumas and D. W. Matula, "Recoders for Partial Compression and Rounding," submitted to 13th *IEEE Symp. Comput. Arithmetic*, 1997.

[Fa 81] P. M. Farmwald, "On the design of high performance digital arithmetic units," in *Ph. D. thesis, Stanford University*, 1981.

[Fe 67] D. Ferrari, "A Division Method Using a Parallel Multiplier," in *IEEE Trans. Electron. Comput.*, 1967, EC-16, pp 224-226.

[IT 95] M. Ito, N. Takagi and S. Yajima, "Efficient Initial Approximation and Fast Converging Methods for Division and Square Root," in *Proc. 12th IEEE Symp. Comput. Arithmetic*, 1995, pp 2-9.

[Ka 91] H. Kadota, "Apparatus For Executing Division By High Speed Convergence Processing," in *United States Patent*, No. 4,991,132, 1991.

[LM 95] A. Lyu and D. W. Matula, "Redundant Binary Booth Recoding", in *Proc. 12th IEEE Symp. Comput. Arithmetic*, 1995, pp 50-57.

[Na 87] H. Nakano, "Method And Apparatus For Division Using Interpolation Approximation," in *United States Patent*, No. 4,707,798, 1987.

[OL 91] W. J. Ooms, C. D. Leitch and R. M. Delgado, "Method And Apparatus For Obtaining The Quotient Of Two Numbers Within One Clock Cycle," in *United States Patent*, No. 5,020,017, 1991.

[SO 93] M.J. Schulte, J. Omar and E.E. Swartzlander, "Optimal Approximations for the Newton-Raphson Division Algorithm," presented at SCAN-93, the *Conference on Scientific Computing, Computer Arithmetic, and Numeric Validation* in Vienna, Sept. 1993.

[WG 95] W. F. Wong and E. Goto, "Fast Evaluation of the Elementary Functions in Single Precision," in *IEEE Trans. Comput.* Vol. 44, No. 3, March 1995.