Computers **2015**, *4*, 293-321; doi:10.3390/computers4040293

OPEN ACCESS

computers

ISSN 2073-431X

www.mdpi.com/journal/computers

Article

Linear and Quadratic Interpolators Using Truncated-Matrix Multipliers and Squarers [†]

E. George Walters III

Penn State Erie, The Behrend College, 5101 Jordan Road, Erie, PA 16563, USA;
E-Mail: waltersg@ieee.org; Tel.: +1-814-898-6390; Fax: +1-814-898-6125

[†] This paper is an extended version of our paper published in [1].

Academic Editor: Pedro Alonso Jordá

Received: 21 August 2015 / Accepted: 10 November 2015 / Published: 16 November 2015

Abstract: This paper presents a technique for designing linear and quadratic interpolators for function approximation using truncated multipliers and squarers. Initial coefficient values are found using a Chebyshev-series approximation and then adjusted through exhaustive simulation to minimize the maximum absolute error of the interpolator output. This technique is suitable for any function and any precision up to 24 bits (IEEE single precision). Designs for linear and quadratic interpolators that implement the $1/x$, $1/\sqrt{x}$, $\log_2(1+2^x)$, $\log_2(x)$ and 2^x functions are presented and analyzed as examples. Results show that a proposed 24-bit interpolator computing $1/x$ with a design specification of ± 1 unit in the last place of the product (ulp) error uses 16.4% less area and 15.3% less power than a comparable standard interpolator with the same error specification. Sixteen-bit linear interpolators for other functions are shown to use up to 17.3% less area and 12.1% less power, and 16-bit quadratic interpolators are shown to use up to 25.8% less area and 24.7% less power.

Keywords: function approximation; linear and quadratic interpolator; truncated multiplier; truncated squarer

1. Introduction

The approximation of functions, such as reciprocal, reciprocal root, logarithm, exponential and others, is important for a wide variety of applications, including general-purpose computing, scientific computing, digital signal processing (DSP) and application-specific processors, such as those used for graphics acceleration. The logarithmic number system (LNS), which greatly reduces the area and power required for multiplication and other operations, requires approximation of a special logarithm to perform addition and subtraction. As transistor densities continue to increase, more of these functions are implemented in hardware. Piecewise-polynomial function approximation, using coefficients stored in a lookup table, is an efficient technique with many papers written on the subject [2–13].

This paper describes the application of truncated-matrix multipliers and squarers to linear and quadratic interpolators to approximate $1/x$, $1/\sqrt{x}$, $\log_2(1 + 2^x)$, $\log_2(x)$ and 2^x. Some of the techniques and results discussed in this paper were presented at the 17th IEEE Symposium on Computer Arithmetic in 2005 [1]. This work is a significant extension of that paper. Section 2 discusses polynomial function approximation and how initial coefficients are obtained using a Chebyshev-series approximation. Section 4 describes general function-interpolator hardware designs that use standard multipliers and squarer and presents an error analysis that is used to determine initial coefficient lengths and rounding precisions. Section 5 describes a method for minimizing the size of the coefficient lookup table, then truncating the partial-product matrices of the arithmetic units to reduce area and power while maintaining the design specifications for error. Section 6 presents results for several interpolator designs using these techniques.

2. Polynomial Function Approximation

Approximating a function $Y = f(X)$ is usually done in three steps. First, range reduction is performed to reduce X to x, where $x \in [x_{min}, x_{max})$. Next, $y = f(x)$ is computed. Finally, $Y = f(X)$ is computed by performing a reconstruction step. Range-reduction and reconstruction techniques for common function approximations are well known [4,13] and are not discussed here.

Polynomial approximations have the following form:

$$\begin{aligned} f(x) &\approx a_0 + a_1x + a_2x^2 + \cdots + a_{N-1}x^{N-1} \\ &\approx \sum_{i=0}^{N-1} a_i x^i \end{aligned} \tag{1}$$

where N is the number of terms in the polynomial approximation, $f(x)$ is the function to be approximated and a_i is the coefficient of the i-th term. In practice, many hardware implementations are limited to linear or quadratic interpolators. For a linear interpolator, $N = 2$, and the approximation is:

$$f(x) \approx a_0 + a_1x \tag{2}$$

For a quadratic interpolator, $N = 3$, and the approximation is:

$$f(x) \approx a_0 + a_1x + a_2x^2 \tag{3}$$

2.1. Investigated Functions

The techniques in this paper are applicable to interpolators for a wide variety of functions. In this work, five functions commonly used in high-performance DSP systems are investigated: reciprocal, reciprocal root, the LNS-addition logarithm, logarithm and exponential. For each of these functions, Table 1 gives the range-reduced function computed by the interpolator, the input and output ranges of the interpolator and the maximum absolute values of the second and third derivatives, $|f''(\xi)|$ and $|f'''(\xi)|$, which are used in the design of the interpolator and its error analysis.

Table 1. Investigated functions. LNS, logarithmic number system.

Function	$f(x)$	**Input Range** $[x_{min}, x_{max})$	**Output Range**	$\|f''(\xi)\|$	$\|f'''(\xi)\|$
Reciprocal	$\frac{1}{x}$	$[1, 2)$	$(0.5, 1]$	2	6
Reciprocal Root					
even exponent	$\frac{1}{\sqrt{x}}$	$[1, 2)$	$(\frac{1}{\sqrt{2}}, 1]$	0.75	1.875
odd exponent	$\frac{1}{\sqrt{x}}$	$[2, 4)$	$(0.5, \frac{1}{\sqrt{2}}]$	0.1326	0.1657
LNS-Addition Logarithm	$\log_2(1 + 2^x)$	$[-(2^{n_{int}}), 0)$	$(0, 1]$	0.1733	0.0462
Logarithm	$\log_2(x)$	$[1, 2)$	$[0, 1)$	1.4427	2.8854
Exponential	2^x	$[0, 1)$	$[1, 2)$	0.9609	0.6660

2.1.1. Reciprocal

The reciprocal function, $f(x) = 1/x$, is often used in DSP systems to implement division efficiently by converting it to a multiplication by a reciprocal. For example, $A \div B$ can be implemented as $A \times 1/B$. The reciprocal can also be used as an initial value for computing higher-precision reciprocals or performing higher-precision division using iterative algorithms, such as Newton–Raphson or the Goldschmidt algorithm [14,15].

2.1.2. Reciprocal Root

The reciprocal root function, $f(x) = 1/\sqrt{x}$, is used in DSP systems to implement division by the square root of a number. One common use for this function is vector normalization. For example, consider the two-dimensional vector $\vec{V}$, which has x-axis and y-axis components of V_i and V_j, respectively. The components of the normalized vector, $\vec{v}$, are computed by:

$$\begin{aligned} v_i &= \frac{V_i}{\sqrt{V_i^2 + V_j^2}} \\ v_j &= \frac{V_j}{\sqrt{V_i^2 + V_j^2}} \end{aligned} \tag{4}$$

These values can be computed efficiently by computing $x = V_i^2 + V_j^2$, then $1/\sqrt{x}$, then:

$$\begin{aligned} v_i &= V_i \cdot 1/\sqrt{x} \\ v_j &= V_j \cdot 1/\sqrt{x} \end{aligned} \tag{5}$$

The reciprocal root function can also be used to compute the square root using the following identity:

$$\sqrt{x} = x \cdot \frac{1}{\sqrt{x}} \tag{6}$$

2.1.3. LNS-Addition Logarithm

In the logarithmic number system, a real number, A, is represented by its sign and the logarithm of its absolute value [16]. Any base can be chosen for the logarithm, but base-two is common and is used exclusively in this work. (In much of the literature on LNS, X and Y are used to represent real numbers, with x and y representing their logarithms. In much of the literature on function interpolation, $Y = f(X)$ is used to represent the value of a function and its argument, with $y = f(x)$ used to represent the value of a function after range reduction has taken place. In order to prevent confusion, X, Y, x and y are used in this paper for function interpolation, while A, B, a and b are used for LNS numbers.) The number A is given as:

$$A = (-1)^{S_A} \cdot 2^a \tag{7}$$

where S_A is the sign of A, with "1" being negative, and the exponent, a, is $\log_2(|A|)$. If $|A| < 1$, then a is negative, so a must be signed. In this work, a is quantized and stored as a two's-complement fixed-point number with K integer bits and F fractional bits. The K integer bits includes the sign bit for a. Counting the sign bit for A, S_A, and the exponent, a, $K + F + 1$ bits are required to represent A.

The range of a is:

$$2^{-K+1} \leq a \leq 2^{K-1} - 2^{-F} \tag{8}$$

so the range of the magnitude of values that can represented is:

$$2^{2^{-K+1}} \leq |A| \leq 2^{2^{K-1}-2^{-F}} \tag{9}$$

The representable numbers adjacent to the representation of A are $2^{a-2^{-F}}$ and $2^{a+2^{-F}}$. From this, it can be seen that the range of representable values is set primarily by K, and F sets the precision.

One significant advantage of LNS is that multiplication and division are simple operations. For multiplication, the exponents of the two operands are simply added. For division, the exponent of the divisor is subtracted from the exponent of the dividend. Squaring and the square root are even simpler. To square a number in LNS, the exponent is multiplied by two, which is easily accomplished by a left shift of one bit. The square root of a number is computed by dividing the exponent by two, which is done by shifting one bit to the right. These operations are summarized in Table 2.

Table 2. Some LNS operations.

Real Operation	LNS Operation	Constraints	LNS Sign	LNS Exponent	Exact?
$A \cdot B$	$((-1)^{S_A} \cdot 2^a) \cdot ((-1)^{S_B} \cdot 2^b)$	none	$S_A \oplus S_B$	$a + b$	yes
$A \div B$	$((-1)^{S_A} \cdot 2^a) \div ((-1)^{S_B} \cdot 2^b)$	none	$S_A \oplus S_B$	$a - b$	yes
A^2	$((-1)^{S_A} \cdot 2^a)^2$	none	0	$2a$	yes
$\sqrt{A}$	$\sqrt{(-1)^{S_A} \cdot 2^a}$	$S_A = 0$	0	$\left\lfloor \frac{a}{2} \right\rfloor$	no

For many DSP algorithms, the majority of silicon area and power are consumed by the operations listed in Table 2, so LNS has great promise. The primary challenge of LNS implementations is that addition and subtraction operations are computationally expensive. As such, optimization of addition and subtraction is an important topic in the LNS literature.

LNS addition of two numbers, A and B, is commonly implemented as [17]:

$$\log_2(|A| + |B|) = \max(a, b) + s_b(-|a - b|) \tag{10}$$

where $s_b(x)$ is known as the LNS-addition logarithm. For base-two LNS,

$$s_b(x) = \log_2(1 + 2^x) \tag{11}$$

which is often implemented by interpolation.

Using Equation (10) for addition ensures that $x \leq 0$, which in turn ensures that $0 \leq s_b(x) \leq 1$. As $x \to -\infty$, $s_b(x) \to 0$. When $s_b(x) < 2^{-F-1}$, it rounds to zero because the smallest representable number is 2^{-F}. This occurs when $x < \log_2(2^{2^{-F-1}} - 1)$ and is known as "essential zero". Interpolator implementations can take advantage of the essential zero with no loss in accuracy by limiting the input range to $x \in [\log_2(2^{2^{-F-1}} - 1), 0]$, and outputting zero for inputs below that range. The logic for detecting the essential zero can be simplified with a small loss in accuracy by recognizing that $\log_2(2^{2^{-F-1}} - 1) \approx -F - 1$.

LNS subtraction can be implemented in a similar manner using the LNS-subtraction logarithm, $d_b(x) = \log_2(1 - 2^x)$. Unfortunately, $d_b(x)$ is difficult to implement efficiently because it approaches $-\infty$ as $x \to 0$. Coleman's transformation [18] mitigates this problem by eliminating the need to evaluate $d_b(x)$ near $x = 0$, so $d_b(x)$ is easier to implement efficiently. However, Coleman's transformation requires an LNS processor to implement both the $s_b(x)$ and the $d_b(x)$ functions. Arnold *et al.*'s transformation [19] eliminates the need to implement both functions by implementing $d_b(x)$ as a function of $s_b(x)$. The drawback is that $s_b(x)$ must accept positive values of x. Arnold's improved cotransformation [20] eliminates this problem, allowing $d_b(x)$ to be implemented as a function of $s_b(x)$ with $x \leq 0$. Vouzis *et al.* [17] improve this cotransformation to work in cases not considered in [20]. Thus, the LNS-addition logarithm investigated in this work can be used for both LNS-addition and LNS-subtraction operations.

2.1.4. Logarithm

The base-two logarithm function, $f(x) = \log_2(x)$, can be used to convert a number in two's-complement representation to LNS representation. It can also be used to compute logarithms for other bases using the identity:

$$\log_y(x) = \frac{\log_2(x)}{\log_2(y)} \tag{12}$$

Note that $1/\log_2(y)$ can be precomputed and stored in memory, so that $\log_y(x)$ can be computed as $\log_2(x)$ multiplied by a constant.

The logarithm function is also important in many classes of graphics-shading programs and is implemented in the instruction sets of modern graphics-processing units (GPUs) [13].

2.1.5. Exponential

The base-two exponential function, $f(x) = 2^x$, can be used to convert a number in LNS representation to two's-complement representation. Like the logarithm, exponentiation is also important for graphics processing and is implemented by modern GPUs.

The logarithm and exponential functions can be used together to compute the powering function, $f(x) = x^y$, using the following identity:

$$\begin{aligned} x^y &= 2^{\log_2(x^y)} \\ x^y &= 2^{y\log_2(x)} \end{aligned} \tag{13}$$

2.2. Partitioning

In order to reduce the order of the polynomial while maintaining the desired output accuracy, the interval $[x_{min}, x_{max})$ is often partitioned into 2^m subintervals of equal size, each with a different set of coefficients. To implement this efficiently in hardware, the interpolator input is split into an m-bit most-significant part, x_m, and an $(n - m)$-bit least-significant part, x_l, where n is the number of bits from x input to the interpolator. If any of the leading bits in x are constants, they are not input to the interpolator, so n is not necessarily the number of bits in x. The value input to the interpolator is $x_m + x_l$, which is a fixed-point number with n_{int} integer bits.

For the LNS-addition logarithm, $x \in [x_{min}, 0)$, where x_{min} varies with the application. As an example, consider the case where $x_{min} = -16$ and $F = 8$. In this example, x has the form "$1xxxx.xxxxxxxx$", and the number of integer bits input to the interpolator, n_{int}, is four. The relationship of x to $x_m + x_l$ is:

$$x = -2^{n_{int}} + x_m + x_l \quad \begin{cases} 0 \leq x_m \leq 2^{n_{int}} - 2^{n_{int}-m} \\ 0 \leq x_l \leq 2^{n_{int}-m} - 2^{n_{int}-n} \end{cases} \tag{14}$$

In the case where $x \in [0, 1)$, such as for the exponential function, x has the form "$0.xxxxxxxx$". The leading "0" is not input to the interpolator, so $n_{int} = 0$, and the relationship of x to $x_m + x_l$ is:

$$x = x_m + x_l \quad \begin{cases} 0 \leq x_m \leq 1 - 2^{-m} \\ 0 \leq x_l \leq 2^{-m} - 2^{-n} \end{cases} \tag{15}$$

In the case where $x \in [1, 2)$, such as for the reciprocal function, x has the form "$1.xxxxxxxx$". The leading "1" is not input to the interpolator, so $n_{int} = 0$, and the relationship of x to $x_m + x_l$ is:

$$x = 1 + x_m + x_l \quad \begin{cases} 0 \leq x_m \leq 1 - 2^{-m} \\ 0 \leq x_l \leq 2^{-m} - 2^{-n} \end{cases} \tag{16}$$

For the reciprocal root function, when the exponent is odd, $x \in [2, 4)$ and has the form "$1x.xxxxxxx$". In this situation, $n_{int} = 1$, and the relationship of x to $x_m + x_l$ is:

$$x = 2 + x_m + x_l \quad \begin{cases} 0 \leq x_m \leq 2 - 2^{-m+1} \\ 0 \leq x_l \leq 2^{-m+1} - 2^{-n+1} \end{cases} \tag{17}$$

2.3. Determining the Partitioning Interval

The coefficients for each subinterval are stored in a lookup table. x_m is used to select the coefficients, so Equation (1) becomes:

$$\begin{aligned} f(x) &\approx a_0(x_m) + a_1(x_m)x_l + a_2(x_m)x_l^2 + \cdots + a_{N-1}(x_m)x_l^{N-1} \\ &\approx \sum_{i=0}^{N-1} a_i(x_m)x_l^i \end{aligned} \tag{18}$$

where $a_0(x_m), a_1(x_m), \ldots, a_{N-1}(x_m)$ are the coefficients.

The approach presented in this paper uses a Chebyshev-series approximation [21] to select the initial coefficient values for each subinterval. These coefficients are then quantized as described in Section 4 and optimized as described in Section 5. Schulte and Swartzlander [4] describe Chebyshev-series approximation in detail. This section summarizes the equations used to generate initial coefficient values for linear and quadratic approximations.

First, the number of subintervals, 2^m, must be determined. When using infinite-precision arithmetic, the error for a function approximated using the Chebyshev-series over the interval $[x_1, x_2)$ is:

$$E_{Chebyshev} = \left(\frac{x_2 - x_1}{4}\right)^N \cdot \frac{2 \cdot |f^N(\xi)|}{N!} \tag{19}$$

where ξ is the point on the interval being approximated, such that the N-th derivative of $f(x)$ is at its maximum absolute value [21]. Table 1 gives these values for the investigated functions.

The range of each subinterval is $[x_m, x_m + 2^{n_{int}-m})$, so Equation (19) becomes:

$$\begin{gathered} E_{Chebyshev} = \frac{2^{-N(m+2-n_{int})+1} \cdot |f^N(\xi)|}{N!} \\ x_m \le \xi < x_m + 2^{n_{int}-m} \end{gathered} \tag{20}$$

The maximum allowable error of the interpolator is defined to be 2^{-q}, which is selected as a design parameter. Error budgets for linear and quadratic designs are discussed in later sections. For both designs, the error budget for $E_{Chebyshev}$ is 1/4 of the total maximum error, so $E_{Chebyshev}$ is limited to 2^{-q-2}, and Equation (20) is solved for m. Since m must be an integer, this gives:

$$m = \left\lceil \frac{q + 3 - 2N + n_{int}N + \log_2(|f^N(\xi)|) - \log_2(N!)}{N} \right\rceil \tag{21}$$

2.4. Calculating the Coefficients

For a linear Chebyshev interpolator, the coefficients used in Equation (18) for the interval $[x_m, x_m + 2^{n_{int}-m})$ are given by:

$$a_0(x_m) = -\frac{1}{2}y_0\left(\sqrt{2} - 1\right) + \frac{1}{2}y_1\left(\sqrt{2} + 1\right) \tag{22}$$

$$a_1(x_m) = \sqrt{2}\,(y_0 - y_1) \cdot 2^{m-n_{int}} \tag{23}$$

where:

$$y_0 = f\left(x_m + \left(2+\sqrt{2}\right)\cdot 2^{n_{int}-m-2}\right) \tag{24}$$

$$y_1 = f\left(x_m + \left(2-\sqrt{2}\right)\cdot 2^{n_{int}-m-2}\right) \tag{25}$$

For a quadratic Chebyshev interpolator, the coefficients for the interval $[x_m, x_m + 2^{n_{int}-m})$ are given by:

$$a_0(x_m) = \frac{1}{3}y_0\left(2-\sqrt{3}\right) - \frac{1}{3}y_1 + \frac{1}{3}y_2\left(\sqrt{3}+2\right) \tag{26}$$

$$\begin{aligned} a_1(x_m) &= \frac{1}{6}y_0\left(\sqrt{3}-4\right)\cdot 2^{m-n_{int}+2} + \frac{1}{3}y_1\cdot 2^{m-n_{int}+4} \\ &\quad - \frac{1}{6}y_2\left(\sqrt{3}+4\right)\cdot 2^{m-n_{int}+2} \end{aligned} \tag{27}$$

$$a_2(x_m) = \frac{1}{3}\left(y_0 - 2y_1 + y_2\right)\cdot 2^{2(m-n_{int})+3} \tag{28}$$

where:

$$y_0 = f\left(x_m + \left(\frac{\sqrt{3}}{2}+1\right)\cdot 2^{n_{int}-m-1}\right) \tag{29}$$

$$y_1 = f\left(x_m + 2^{n_{int}-m-1}\right) \tag{30}$$

$$y_2 = f\left(x_m + \left(1-\frac{\sqrt{3}}{2}\right)\cdot 2^{n_{int}-m-1}\right) \tag{31}$$

Coefficients for cubic Chebyshev interpolators are given in [22]. Coefficients for higher-order Chebyshev interpolators can be computed using the equations presented in [4].

3. Truncated Multipliers and Squarers

Truncated multipliers and squarers are units in which several of the least-significant columns of partial products are not formed [23]. Eliminating partial products from the multiplication or squaring matrix reduces the area of the unit by eliminating the logic needed to generate those terms, as well as reducing the number of adder cells required to reduce the matrix prior to the final addition. Additional area savings are realized because a shorter carry-propagate adder can be used to compute the final results, which often yields reduced delay, as well. Eliminating adder cells and, thus, their related switching activity also results in reduced power consumption.

Figure 1 shows a 14×10-bit truncated multiplier, where r denotes the number of unformed columns and k denotes the number of columns that are formed, but discarded in the final result. In this example, $r = 7$ and $k = 2$. Eliminating partial products introduces a reduction error, E_r, into the output. This error ranges from E_{r_low}, which occurs when each of the unformed partial-product bits is a "1", to E_{r_high}, which occurs when each is a "0". E_{r_low} is given by [24]:

$$E_{r\_low} = -\left((r-1)\cdot 2^r + 1\right)\cdot 2^{-r-k}\text{ ulps} \tag{32}$$

where ulps is units in the last place of the product. Since E_{r_high} is zero, the range of the reduction error is:

$$-\left((r-1)\cdot 2^{r}+1\right)\cdot 2^{-r-k}\ \text{ulps} \le E_r \le 0 \tag{33}$$

In the example given in Figure 1, the range of reduction error is $-1.502\ \text{ulps} \le E_r \le 0\ \text{ulps}$. In comparison, the error due to rounding a product to the nearest has a range of $-0.5\ \text{ulps} < E_{rnd} \le 0.5\ \text{ulps}$.

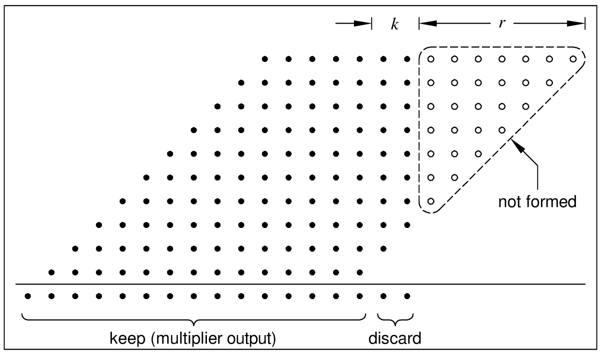


Figure 1. A 14 × 10-bit truncated multiplier, $k = 2$, $r = 7$.

Figure 2 shows a 12-bit truncated squarer. Truncated squarers are an extension of specialized squarers, which are described in [25]. As with truncated multipliers, r denotes the number of unformed columns, and k denotes the number of columns that are formed, but discarded in the final result. Unlike truncated multipliers, it is impossible for each of the unformed partial-product bits in a truncated squarer to be "1". E_{r_low} for a truncated squarer is given by [26]:

$$E_{r\_low} = \begin{cases} -\left(2^{r-1}\cdot r - 2^{r}+1\right) \\ \quad \cdot 2^{-r-k}\ \text{ulps}, \ \text{if } r \text{ is even} \\ -\left(2^{r-1}\cdot r - \dfrac{11}{8}\cdot 2^{r} + 2^{\left(\frac{1}{2}r-\frac{1}{2}\right)}+1\right) \\ \quad \cdot 2^{-r-k}\ \text{ulps}, \ \text{if } r \text{ is odd} \end{cases} \tag{34}$$

The range of reduction error for a truncated squarer is $E_{r\_low} \le E_r \le 0$. In the example given in Figure 2, $k = 2$, $r = 10$, and the range of reduction error is $-1.000\ \text{ulps} \le E_r \le 0\ \text{ulps}$.

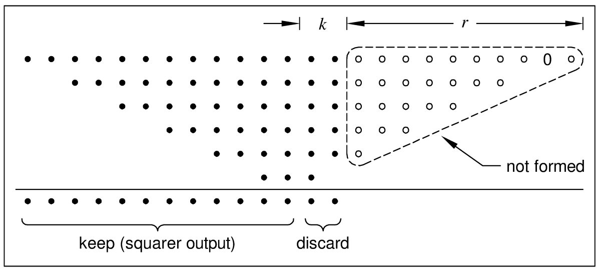


Figure 2. A 12-bit truncated squarer, $k = 2$, $r = 10$.

For generality, k and r will be used in equations given in later sections to describe both standard and truncated multipliers and squarers. For a standard unit, $r = 0$, because all columns of partial products are formed.

3.1. Constant Correction

From the previous discussion, it can be seen that the reduction error for both truncated multipliers and truncated squarers is always negative. One way to offset this error is to add a constant to the partial-product matrix [24]. In some applications, it is desirable to select a constant that makes the average error of the multiplier or squarer as close to zero as possible. When designing a function interpolator, however, it is usually desirable to minimize the maximum absolute error. This is done by selecting a correction constant, C_{cc_abs}, equal to the additive inverse of the midpoint of the range of the error. This value is:

$$C_{cc\_abs} = -\frac{E_{r\_low}}{2} \tag{35}$$

where E_{r_low} is given by Equation (32) for truncated multipliers and Equation (34) for truncated squarers. In practice, C_{cc_abs} is rounded, so that it does not have any bits in the r least-significant columns, since those bits will have no effect on the final output.

3.2. Variable Correction

Another way to offset reduction error is through variable correction [27]. Figure 3 shows a truncated squarer with variable correction. With this technique, the most-significant column of unformed partial products is added to the next most-significant column. This can be thought of as rounding each row of partial products, such that the r least-significant columns are eliminated.

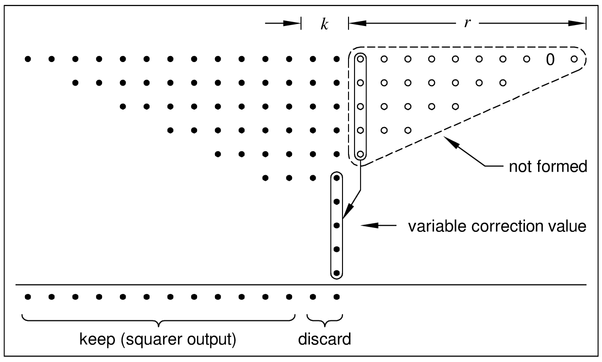


Figure 3. A 12-bit truncated squarer with variable correction.

4. Preliminary Hardware Design

4.1. Finite-Precision Arithmetic Effects

In addition to the error of the Chebyshev approximation, there are errors due to quantization effects, multiplier/squarer rounding and rounding of the final addition that affect the accuracy of the interpolator output.

Each coefficient a_i is quantized to n_i bits using the round-to-nearest approach and stored in a lookup table. The least-significant bit of each coefficient has a weight of $2^{-n_{fi}}$, where n_{fi} is the number of fractional bits in coefficient a_i. The difference between a quantized coefficient and its infinite-precision value is defined as ε_i, where $|\varepsilon_i| \leq 2^{-n_{fi}-1}$.

In order to prevent excessive intermediate word lengths, multiplier and squarer outputs are rounded. When standard multipliers are used, rounding is accomplished by adding a "1" to the column immediately to the right of the rounding point, then truncating the k least-significant bits at the output. Recall that k is the number of columns in the multiplication matrix that are formed and added, but discarded in the final result. The maximum rounding error, E_{rnd}, is 0.5 ulps, where 1 ulp is the weight of the least-significant bit output by the multiplier or squarer.

4.2. Linear Interpolator

Figure 4 shows the block diagram of a linear interpolator that computes Equation (2). x_m is used to select coefficients a_0 and a_1 from a lookup table. Multiplier 1 computes $a_1 \cdot x_l$, which is then added to a_0 to produce the output. The multiplier output can be kept in carry-save form or a multiply-accumulate (MAC) unit may be used to reduce the overall area and delay of the interpolator.

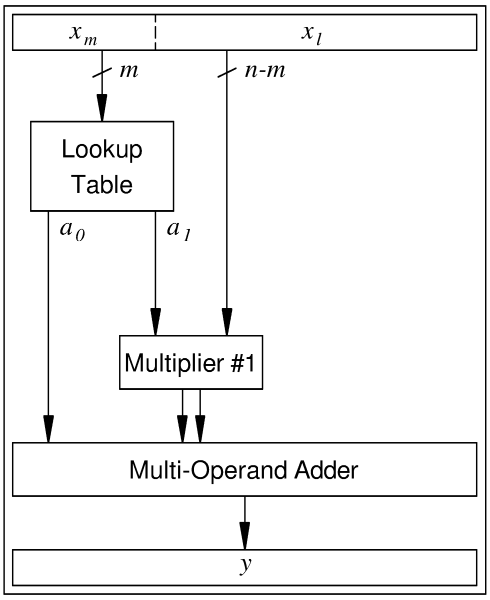


Figure 4. Linear interpolator block diagram.

Table 3 gives the error budget for the initial linear-interpolator design. As described earlier, the maximum allowable error for the interpolator is defined to be 2^{-q}, which is selected as a design parameter. For the Chebyshev approximation, m is selected to limit the approximation error to 1/4 of the total error. The total error budget is slightly more than the maximum allowable error, 2^{-q}. However, parameters are selected for each source of error, such that the maximum error is less than or equal to the budget amount. Therefore, it is unlikely that each source of error would be at its maximum value at the same time, so these budget values have been found to work for most designs. When these values result in an initial design that does not meet the error specification, one or more of the design parameters are tightened to bring the design into compliance.

Table 3. Error budget for linear interpolators.

Source of Error	**Error Budget**	
	Absolute	**Fraction**
Chebyshev approximation	2^{-q-2}	$1/4$
Coefficient quantization (2^{-q-3} each)	2^{-q-2}	$1/4$
Multiplier rounding	2^{-q-4}	$1/16$
Rounding at interpolator output	2^{-q-1}	$1/2$
Total	$\frac{17}{16} \cdot 2^{-q}$	$17/16$

Errors in the output due to the quantization of a_0 and a_1 are E_{ε_0} and E_{ε_1}, respectively. Since a_0 contributes directly to the output, $E_{\varepsilon_0} = \varepsilon_0$, so:

$$|E_{\varepsilon_0}| \leq 2^{-n_{f0}-1} \tag{36}$$

a_1 is multiplied by x_l, which has a maximum value less than $2^{n_{int}-m}$, so:

$$|E_{\varepsilon_1}| < 2^{-n_{f1}-1} \cdot 2^{n_{int}-m} \tag{37}$$

The error budget for quantization of each coefficient is 1/8 of the total error. Thus, the coefficient length is determined by setting $E_{\varepsilon_i} = 2^{-q-3}$ and solving for n_{fi}, so:

$$\begin{aligned} E_{\varepsilon_0} &= 2^{-q-3} \\ 2^{-n_{f0}-1} &= 2^{-q-3} \\ n_{f0} &= q+2 \end{aligned} \tag{38}$$

and:

$$\begin{aligned} E_{\varepsilon_1} &= 2^{-q-3} \\ 2^{-n_{f1}-1} \cdot 2^{n_{int}-m} &= 2^{-q-3} \\ n_{f1} &= q + n_{int} - m + 2 \end{aligned} \tag{39}$$

Section 5 describes how these lengths are then reduced to the minimum precision that maintains the desired accuracy of the interpolator. In addition to the fractional bits, a sign bit is needed, so:

$$n_0 = n_{f0} + 1 = q + 3 \tag{40}$$

$$n_1 = n_{f1} + 1 = q + n_{int} - m + 3 \tag{41}$$

If additional bits to the left of the binary point are required (e.g., if any value of $|a_i|$ is greater than or equal to one), these values are incremented accordingly. Likewise, if some of the most-significant fractional bits are always constant, these values are decremented accordingly. For example, if all values of a_i are less than 0.25, the two most-significant fractional bits are always zero and do not need to be stored in the coefficient table or input to the multiplier.

In addition to quantization errors, rounding the multiplier output introduces a rounding error, E_{rnd_m1}, at the interpolator output. The least-significant-bit (LSB) weight of a_1 is $2^{-n_{f1}}$, and the LSB weight of x_l is $2^{n_{int}-n}$, so the LSB weight of a full-precision product would be $2^{-n_{f1}+n_{int}-n}$. Since r columns of partial products are not formed and k output bits are discarded, the LSB weight of the multiplier output is $2^{-n_{f1}+n_{int}-n+k_{m1}+r_{m1}}$, so:

$$E_{rnd\_m1} = 2^{-n_{f1}+n_{int}-n+k_{m1}+r_{m1}-1} \tag{42}$$

where k_{m1} and r_{m1} are k and r for the multiplier.

The initial design for linear interpolation uses a standard multiplier. The error budget for multiplier rounding is 1/16 of the total error, so k_{m1} is chosen by setting $E_{rnd\_m1} = 2^{-q-4}$ and solving for k_{m1}.

$$\begin{aligned} E_{rnd\_m1} &= 2^{-q-4} \\ 2^{-n_{f1}+n_{int}-n+k_{m1}+r_{m1}-1} &= 2^{-q-4} \\ k_{m1} &= -q + n_{f1} - n_{int} + n - r_{m1} - 3 \end{aligned} \tag{43}$$

For a standard multiplier, all columns of partial products are formed, so $r_{m1} = 0$. Substituting Equation (39) for n_{f1} gives:

$$k_{m1} = n - m - 1 \tag{44}$$

The equations for finding the initial design parameters for linear interpolators are summarized in Table 4.

Table 4. Initial design parameters for linear interpolators.

Design Parameter	Initial Value	Description
n_0	$q + 3$	Length of coefficient a_0, assuming $\lvert a_0\rvert < 1$
n_1	$q + n_{int} - m + 3$	Length of coefficient a_1, assuming $\lvert a_1\rvert < 1$
k_{m1}	$n - m - 1$	Multiplier 1, number of columns formed, but discarded in the final result

4.3. Quadratic Interpolator

Figure 5 shows the block diagram of a quadratic interpolator that computes Equation (3). x_m is used to select coefficients a_0, a_1 and a_2 from a lookup table. A specialized squarer computes x_l^2. Multiplier 1 computes $a_1 \cdot x_l$, and Multiplier 2 computes $a_2 \cdot x_l^2$, both of which are then added to a_0 to produce the output. As with the linear interpolator, the multiplier outputs can be kept in carry-save form.

E_{ε_0} and E_{ε_1} for a quadratic interpolator are the same as for a linear interpolator, given by Equations (36) and (37). a_2 is multiplied by the squarer output, which has a maximum value less than $2^{2(n_{int}-m)}$, so:

$$|E_{\varepsilon_2}| < 2^{-n_{f2}-1} \cdot 2^{2(n_{int}-m)} \tag{45}$$

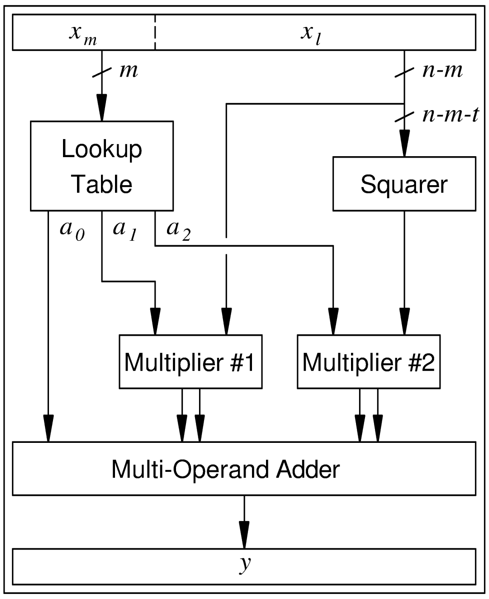


Figure 5. Quadratic interpolator block diagram.

Table 5 gives the error budget for the initial quadratic-interpolator design. The error budget for quantization of each coefficient is 1/16 of the total error, so the coefficient length is determined by setting E_{ε_i} equal to 2^{-q-4} and solving for n_{fi}, giving:

$$\begin{aligned} E_{\varepsilon_0} &= 2^{-q-4} \\ 2^{-n_{f0}-1} &= 2^{-q-4} \\ n_{f0} &= q+3 \end{aligned} \tag{46}$$

$$\begin{aligned} E_{\varepsilon_1} &= 2^{-q-4} \\ 2^{-n_{f1}-1} \cdot 2^{n_{int}-m} &= 2^{-q-4} \\ n_{f1} &= q+n_{int}-m+3 \end{aligned} \tag{47}$$

and:

$$\begin{aligned} E_{\varepsilon_2} &= 2^{-q-4} \\ 2^{-n_{f2}-1} \cdot 2^{2(n_{int}-m)} &= 2^{-q-4} \\ n_{f2} &= q+2n_{int}-2m+3 \end{aligned} \tag{48}$$

Assuming a sign bit in addition to the fractional bits,

$$n_0 = n_{f0}+1 = q+4 \tag{49}$$

$$n_1 = n_{f1}+1 = q+n_{int}-m+4 \tag{50}$$

$$n_2 = n_{f2}+1 = q+2n_{int}-2m+4 \tag{51}$$

As noted above, these values are incremented accordingly if additional bits to the left of the binary point are required, or decremented if all values of $|a_i|$ are less than 0.5.

Table 5. Error budget for quadratic interpolators.

Source of Error	**Error Budget**	
	Absolute	**Fraction**
Chebyshev approximation	2^{-q-2}	$1/4$
Coefficient quantization (2^{-q-4} each)	$3 \cdot 2^{-q-4}$	$3/16$
Truncation of squarer input (x'_l)	2^{-q-4}	$1/16$
Multiplier and squarer rounding (2^{-q-5} each)	$3 \cdot 2^{-q-5}$	$3/32$
Rounding at interpolator output	2^{-q-1}	$1/2$
Total	$\frac{35}{32} \cdot 2^{-q}$	$35/32$

Analysis shows that for some configurations, x_l, can be truncated at the input to the squarer to reduce the size of the squarer. Assume that t least-significant bits of x_l are truncated, such that $x_l = x'_l + \varepsilon_{x_l}$, where x'_l is the truncated version of x_l. The squarer output is then ${x'_l}^2$, rather than x_l^2, resulting in a squarer output error of $-2x'_l \cdot \varepsilon_{x_l} - \varepsilon_{x_l}^2$. Noting that $x'_l < 2^{n_{int}-m}$, $|\varepsilon_{x_l}| < 2^{n_{int}-n+t}$ and $\varepsilon_{x_l}^2$ is negligible, the magnitude of the squarer output error is less than $2^{2n_{int}-n-m+t+1}$. This error is then multiplied by a_2, so the error at the interpolator output due to ε_{x_l} is:

$$|E_{\varepsilon_{x_l}}| < 2^{2n_{int}-n-m+t+1} \tag{52}$$

assuming all values of $|a_2|$ are less than or equal to one. The error budget for truncation of the squarer input is 1/16 of the total error, so $E_{\varepsilon_{x_l}}$ is set equal to 2^{-q-4} to find the maximum value for t, which gives:

$$\begin{aligned} E_{\varepsilon_{x_l}} &= 2^{-q-4} \\ 2^{2n_{int}-n-m+t+1} &= 2^{-q-4} \\ t &= -q - 2n_{int} + n + m - 5 \end{aligned} \tag{53}$$

If any value of $|a_2|$ is larger than one, or if all values of $|a_2|$ are less than 0.5, $|E_{\varepsilon_{x_l}}|$ varies accordingly, which affects the calculation for t. For every additional bit required to represent a_2, t decreases by one. If all values of $|a_2|$ are less than 0.5, then one or more of the most-significant fractional bits are always zero. For each of those bits, t can be increased by one. For example, if all values of $|a_2|$ are less than 0.25, two of the most-significant fractional bits are always zero, so t can be increased by two. If $t \leq 0$, x_l cannot be truncated.

As with the linear interpolator, k is set for the multipliers and the squarer to limit the rounding error. The error budget for rounding is 1/32 of the total error for each unit, so E_{rnd} is set equal to 2^{-q-5}, and the equation is solved for k.

The LSB weight of the Multiplier 1 output is $2^{-n_{f1}+n_{int}-n+k_{m1}+r_{m1}}$, so:

$$E_{rnd\_m1} = 2^{-n_{f1}+n_{int}-n+k_{m1}+r_{m1}-1} \tag{54}$$

Setting this equal to 2^{-q-5} and solving for k_{m1} gives:

$$\begin{aligned} E_{rnd\_m1} &= 2^{-q-5} \\ 2^{-n_{f1}+n_{int}-n+k_{m1}+r_{m1}-1} &= 2^{-q-5} \\ k_{m1} &= -q + n_{f1} - n_{int} + n - r_{m1} - 4 \end{aligned} \tag{55}$$

Standard multipliers are used for the initial design, so $r_{m1} = 0$. Substituting Equation (47) for n_{f1} gives:

$$k_{m1} = n - m - 1 \tag{56}$$

The LSB weight of the squarer output is $2^{2n_{int}-2n+2t+k_{sq}+r_{sq}}$, so:

$$E_{rnd\_sq} = 2^{2n_{int}-2n+2t+k_{sq}+r_{sq}-1} \tag{57}$$

where k_{sq} and r_{sq} are k and r for the squarer. The squarer output is multiplied by a_2, which is assumed to be less than or equal to one, so the rounding error for the squarer is set equal to 2^{-q-5} to find k_{sq}:

$$\begin{aligned} E_{rnd\_sq} &= 2^{-q-5} \\ 2^{2n_{int}-2n+2t+k_{sq}+r_{sq}-1} &= 2^{-q-5} \\ k_{sq} &= -q - 2n_{int} + 2n - 2t - r_{sq} - 4 \end{aligned} \tag{58}$$

A standard squarer is used for the initial design, so $r_{sq} = 0$, so:

$$k_{sq} = -q - 2n_{int} + 2n - 2t - 4 \tag{59}$$

If any values of $|a_2|$ are greater than 1.0, k_{sq} is decreased accordingly, and if all values of $|a_2|$ are less than 0.5, k_{sq} is increased accordingly.

The LSB weight of the Multiplier 2 output equals the LSB weight of a_2 multiplied by the LSB weight of the squarer output, which is $2^{-n_{f2}+2n_{int}-2n+2t+k_{sq}+r_{sq}+k_{m2}+r_{m2}}$, so:

$$E_{rnd\_m2} = 2^{-n_{f2}+2n_{int}-2n+2t+k_{sq}+r_{sq}+k_{m2}+r_{m2}-1} \tag{60}$$

Setting the maximum rounding error equal to 2^{-q-5} and solving for k_{m2} gives:

$$\begin{aligned} &E_{rnd\_m2} = 2^{-q-5} \\ &2^{-n_{f2}+2n_{int}-2n+2t+k_{sq}+r_{sq}+k_{m2}+r_{m2}-1} = 2^{-q-5} \\ &k_{m2} = -q + n_{f2} - 2n_{int} + 2n - 2t - k_{sq} - r_{sq} - r_{m2} - 4 \end{aligned} \tag{61}$$

Substituting Equation (48) for n_{f2} gives:

$$k_{m2} = 2n - 2m - 2t - k_{sq} - r_{sq} - r_{m2} - 1 \tag{62}$$

Standard multipliers and squarers are used for the initial design, so $r_{sq} = 0$ and $r_{m2} = 0$. Substituting Equation (58) for k_{sq} gives:

$$k_{m2} = q + 2n_{int} - 2m + 3 \tag{63}$$

Because k_{sq} is subtracted, the range of values for $|a_2|$ affects the calculation. If any values of $|a_2|$ are greater than 1.0, k_{m2} is increased accordingly, and if all values of $|a_2|$ are less than 0.5, k_{m2} is decreased accordingly.

The equations for finding the initial design parameters for quadratic interpolators are summarized in Table 6.

Table 6. Initial design parameters for quadratic interpolators.

Design Parameter	Initial Value	Description
n_0	$q + 4$	Length of coefficient a_0, assuming $\|a_0\| < 1$
n_1	$q + n_{int} - m + 4$	Length of coefficient a_1, assuming $\|a_1\| < 1$
n_2	$q + 2n_{int} - 2m + 4$	Length of coefficient a_2, assuming $\|a_2\| < 1$
k_{m1}	$n - m - 1$	Multiplier 1, number of columns formed, but discarded in the final result
k_{m2}	$q + 2n_{int} - 2m + 3$	Multiplier 2, number of columns formed, but discarded in the final result
k_{sq}	$-q - 2n_{int} + 2n - 2t - 4$	Squarer, number of columns formed, but discarded in the final result
t	$-q - 2n_{int} + n + m - 5$	Number of bits in x_l that are not input to the squarer

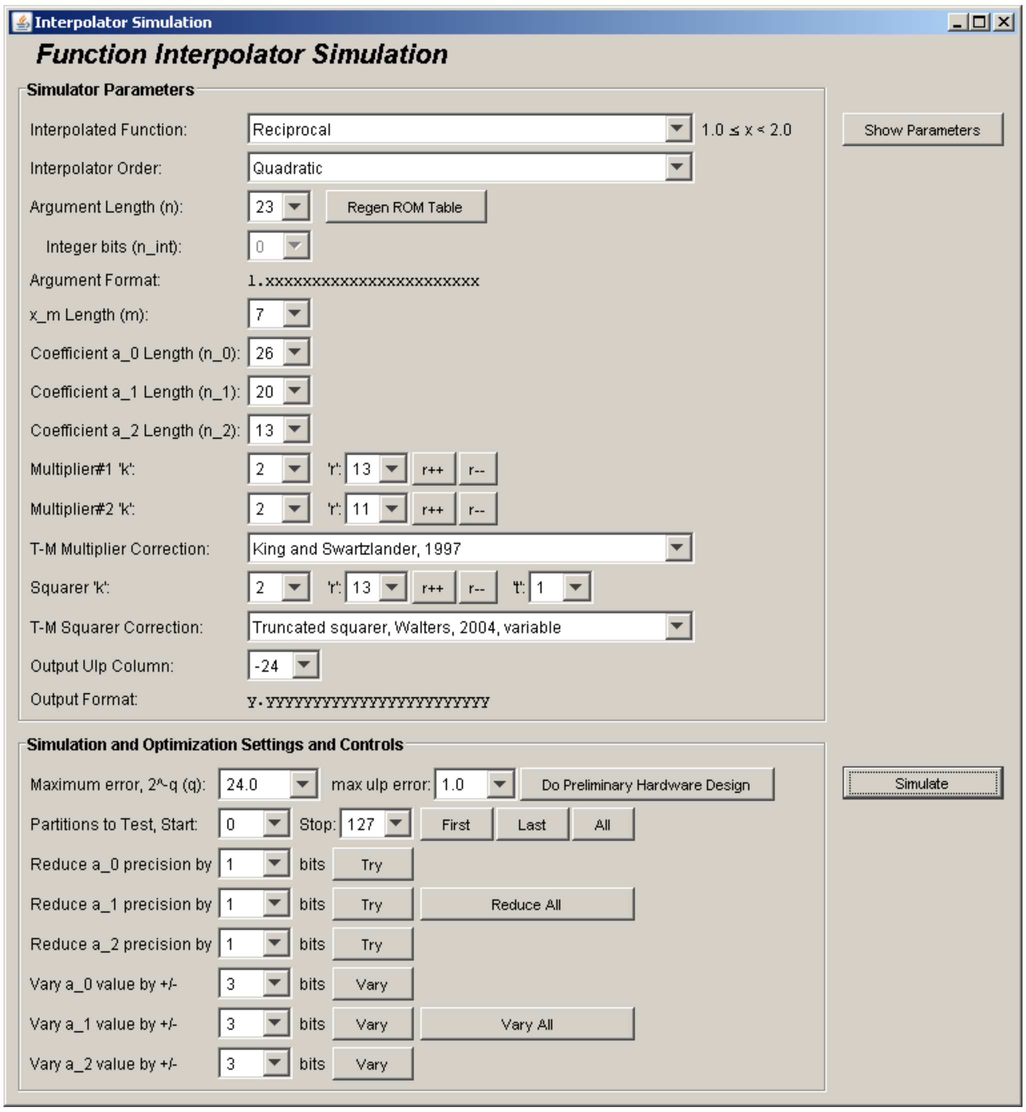
5. Design Optimization

After the preliminary design is complete, the coefficients are optimized through exhaustive simulation and coefficient adjustment. First, simulation is done using standard multipliers and squarers to find the minimum coefficient lengths that can be used while maintaining a maximum absolute output error less than 2^{-q}. This reduces the lookup table size. Next, the standard multipliers and squarers are replaced with truncated-matrix units, and simulation is done to maximize the number of unformed columns while remaining within design specifications for error. This reduces the area and power consumption of the interpolator unit.

5.1. Simulation Software

Exhaustive simulation is performed using a software tool based on an algorithm for fast, bit-accurate simulation of truncated-matrix multipliers and squarers [28]. An interactive graphical user interface (GUI) is provided that allows the user to change the design parameters, then to perform the simulation of all possible input values on some or all of the 2^m subintervals. Figure 6 shows a screenshot. The simulation is bit-accurate, taking into account rounding effects of table lookups and arithmetic units, as well as accounting for the reduction error and correction techniques of truncated-matrix multipliers and squarers. The software also includes a routine for adjusting the coefficients to minimize the maximum absolute error of the output. Simplified pseudocode for this routine is as follows:

```
i = 0
while (i < N)
  for delta = -3 to +3 ulps
    simulate exhaustively, using a(i) + delta
    if error is improved, set a(i) = a(i) + delta
  next delta
  if a(i) was changed
    i = 0
  else
    i++
  end if
end while
```


Interpolator Simulation

Function Interpolator Simulation

Simulator Parameters

Interpolated Function: Reciprocal 1.0 ≤ x < 2.0

Interpolator Order: Quadratic

Argument Length (n): 23

Integer bits (n_int): 0

Argument Format: 1.xxxxxxxxxxxxxxxxxxxxxxx

x_m Length (m): 7

Coefficient a_0 Length (n_0): 26

Coefficient a_1 Length (n_1): 20

Coefficient a_2 Length (n_2): 13

Multiplier#1 'k': 2 'r': 13

Multiplier#2 'k': 2 'r': 11

T-M Multiplier Correction: King and Swartzlander, 1997

Squarer 'k': 2 'r': 13 't': 1

T-M Squarer Correction: Truncated squarer, Walters, 2004, variable

Output Ulp Column: -24

Output Format: y.yyyyyyyyyyyyyyyyyyyyyyyy

Simulation and Optimization Settings and Controls

Maximum error, 2^-q (q): 24.0 max ulp error: 1.0

Partitions to Test, Start: 0 Stop: 127

Reduce a_0 precision by 1 bits

Reduce a_1 precision by 1 bits

Reduce a_2 precision by 1 bits

Vary a_0 value by +/- 3 bits

Vary a_1 value by +/- 3 bits

Vary a_2 value by +/- 3 bits

Figure 6. Screenshot of the function-interpolator simulation GUI.

This method for adjusting coefficients is similar to that presented by Schulte and Swartzlander [4]. One difference is that the coefficient a_i is varied by ± 3 ulps, rather than ± 1 ulp, because it is possible for increasing/decreasing values of a_i to produce identical or worse error results before a better result is achieved, due to the non-monotonic nature of truncated-matrix arithmetic units. Another difference is that if a_i changes, i is reset to zero to readjust all lower-order coefficients before higher-order coefficients are adjusted. Thus, priority for adjustment goes in the order $a_0, a_1, \ldots, a_{N-1}$.

The software performs a large number of simulations, so there are practical limits to the size of the interpolator that can be simulated. For this research, the program is run on a laptop computer with a 1.86-GHz Pentium M processor. The time required to run the coefficient-optimization routine for a 24-bit quadratic interpolator is on the order of five to fifteen minutes. This must be done every time a parameter is changed to see if the new configuration meets the design specifications, so the time required to optimize the design is on the order of several hours. Since the software is developed for research purposes, rather than production purposes, there is a great deal of room for improvement in execution speed. For example, the code could be easily ported to C++, optimized, and run on a faster machine. Exhaustive simulation lends itself well to parallel processing, offering another approach to improve speed. Considering the time and money involved in developing commercial hardware that could benefit from this approach, a few hours of computer time is probably insignificant.

5.2. Optimization Using Standard Multipliers and Squarers

Using the simulation software just described, the initial hardware design is tested and adjusted to reduce the coefficient lengths as much as possible while maintaining the desired accuracy. This is done through trial-and-error by reducing a single coefficient length by one bit, then running the coefficient-optimization routine described above. If the resulting output error range is acceptable, the length of another coefficient is reduced by one bit. If the error is not acceptable, the coefficient is restored to its previous length. This process continues until each coefficient is reduced to a minimum length, typically one or two bits less than the initial lengths. At this point, the size of the lookup table is minimized. Next, multiplier and squarer output lengths are minimized by attempting to increase k_{m1}, k_{m2} and k_{sq} through trial-and-error, as is done for the coefficient lengths, resulting in the final design for a standard interpolator.

If desired, the software allows the user to further explore the design space. Increasing the precision of the multipliers and the squarer by decreasing k below the originally calculated values may allow one or more coefficient lengths to be further reduced, at the expense of larger arithmetic units. The flexibility of the software allows such design trade-offs to be easily quantified.

5.3. Optimization Using Truncated-Matrix Units

After the size of the lookup table is minimized as described in the previous section, the multipliers and the squarer are replaced with truncated-matrix units. The goal is to maximize the number of unformed columns of partial-product bits in each unit without exceeding the interpolator output error bounds. This reduces the area and power consumption of the interpolator.

As with finding the minimum coefficient lengths, the maximum value of r for each unit is obtained by trial-and-error. Note that for a standard multiplier or squarer, $r = 0$, because all partial products are formed. As r is increased for a truncated-matrix unit, k must be decreased by an equal amount in order to maintain the precision and weight of the output product or square.

A good place to start when using truncated-matrix units with constant correction is to set $k = 4$ and $r = k_{std} - 4$, where k_{std} is the value of k used for the original standard multiplier or squarer. For variable-correction units, which are generally more accurate for equivalent values of r, one should start by setting $k = 3$ and $r = k_{std} - 3$. The coefficient-optimization routine is run each time k and r are changed, and the error bounds are determined through exhaustive simulation. In some designs, r cannot be set very high without violating error constraints. In these cases, decreasing k by one often allows r to be set much higher, with the large decrease in matrix size more than offsetting the effect of the increase in output size.

As with coefficient length reduction described above, k is increased and r is decreased for each unit until coefficient optimization fails to produce an acceptable range of output error. At this point, the previously acceptable values for k and r are restored, and the process continues until the maximum number of unformed columns is determined for each unit. When this is complete, the final truncated-matrix interpolator design is found.

5.4. Interpolator Error

Figure 7 shows an example of interpolator error. In each graph, the true value of an arbitrary function $f(x)$ is shown over a small range by a continuous line. The tick marks on the x-axis represent the possible inputs to the interpolator, and the tick marks on the y-axis represent the possible outputs, each tick mark being one ulp.

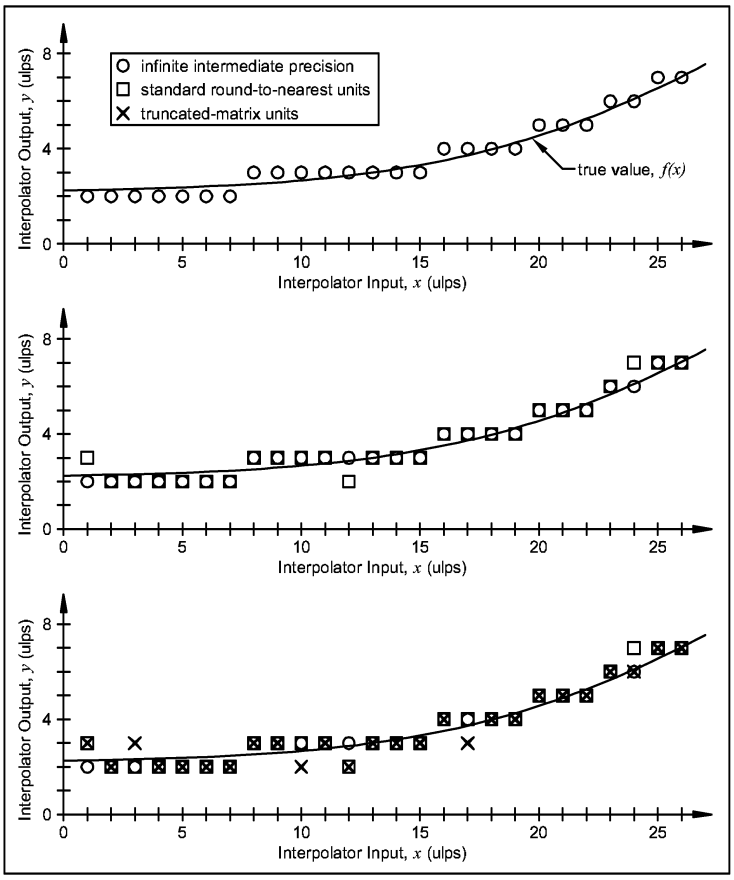


Figure 7. Interpolator error example.

The top graph shows the ideal outputs of a digital function interpolator, indicated by circles. Using infinite precision for intermediate calculations and true round-to-nearest at the output, the output error is limited to $\pm\frac{1}{2}$ ulp. For any function, a $\pm\frac{1}{2}$ ulp error can be achieved using finite-precision intermediate calculations [4], but for some functions, this may result in long intermediate-precision word lengths and a large lookup table.

For many applications, errors larger than $\pm\frac{1}{2}$ ulp can be tolerated [9]. The middle graph of Figure 7 adds squares to indicate the output of an interpolator that uses standard multipliers and squarers, optimized as described in Section 5.2 for a maximum error less than ±1 ulp. Input values of 1, 12 and 24 are shown to have outputs that differ from the ideal values, but still within a ±1 ulp error. Note that keeping the error less than ±1 ulp is also known as faithful rounding [29].

The bottom graph of Figure 7 adds "$\times$"s to indicate the output of a truncated-matrix interpolator that is optimized as described in Section 5.3. When r is zero for each multiplier and squarer in a truncated-matrix interpolator, it is identical to a standard round-to-nearest interpolator and has the same output values. As r is increased, the resulting errors cause some outputs to change. In the bottom graph, the input values of 3, 10, 17 and 24 produce outputs that differ from the standard interpolator. For the first three of those, the error increases, but for $f(24)$, the error actually decreases, because the errors due to finite-precision arithmetic and the errors of the truncated-matrix units are opposite and partially cancel out. This explains why a standard interpolator with less than ± 1 ulp error can tolerate additional errors due to using truncated-matrix multipliers and squarers and still have less than a ± 1 ulp error.

6. Experimental Results

6.1. Methodology

In order to verify the techniques described in this paper and to quantify the benefits, a number of interpolator designs are developed and synthesized. First, linear and quadratic interpolators for the reciprocal function are created for several bit widths. Next, 16-bit linear and quadratic interpolators are designed for the other functions given in Table 1.

For each of the designs, the maximum error is specified to be less than ± 1 ulp. The designs are simulated using the bit-accurate simulation tools described in Section 5.1 to ensure that they meet the error specifications. Verilog models for each design are generated using the Verilog-generation tools that have evolved from work presented in [30]. For a given design, the constant-correction and variable-correction truncated-matrix versions use the same size of lookup table as the standard round-to-nearest interpolators, but with slightly different values. Because the tables are so similar and can be implemented in different ways, the lookup tables are not included in the model, just the arithmetic units and the pipeline registers. Parallel-tree multipliers and squarers using reduced-area reduction are employed, and the multi-operand adder also uses a parallel-tree structure with reduced-area reduction [31]. Otherwise, the models match the block diagrams given in Figures 4 and 5. The models are pipelined in three stages, with the lookup table (and squarer for quadratic designs) in the first stage, the multiplier(s) in the second stage and the multi-operand adder in the final stage.

The designs are synthesized using Synopsys Design Compiler and the Taiwan Semiconductor Manufacturing Company (TSMC) tcbn65gplustc 65-nm standard-cell library with nominal-case operating conditions of $25\,^{\circ}\mathrm{C}$ and $V_{DD} = 1.0\ V$. Timing and area constraints for linear interpolators are set to 0.45 ns and 1 $\mu\mathrm{m}^2$, respectively, to find the smallest designs that can run at that speed. A worst-case delay of 0.45 ns is chosen because it pushes the limits of how fast each design can run, yet each design can meet that speed requirement. Synthesizing each design at the same speed provides a more accurate comparison of area and power for each model. Timing and area constraints for quadratic interpolators are set to 0.60 ns and 1 $\mu\mathrm{m}^2$, respectively, for the same reasons.

6.2. Improvements in Area, Delay and Power

Table 7 shows baseline designs using standard multipliers and squarers. The number of address lines for the coefficient lookup table are given, as well as the total size of the table in bits. Lookup table sizes are comparable to other interpolator design techniques [11,13]. Estimates for area, delay and power are given. Delay is reported in nanoseconds, as well as in fanout-of-four (FO4) delay, which is 31.3 picoseconds for the TSMC library that is used for synthesis. As one would expect, area and power increase as the number of input bits increases. Furthermore, the arithmetic unit portion of quadratic interpolators requires more area and power than for linear interpolators of the same input size. As mentioned above, the area, delay and power shown in Table 7 do not include the lookup table.

Table 7. Baseline interpolators using standard multipliers. FO4, fanout-of-four.

Input Bits	Type	Function	Coefficient Table Address Lines	Coefficient Table Size (bits)	Area (μm^2)	Delay (ns)	Delay (FO4)	Power (mW)
12	Linear	Reciprocal	5	704	1925	0.450	14.37	4.375
16	Linear	Reciprocal	8	6400	2331	0.453	14.46	5.203
20	Linear	Reciprocal	10	31,744	3449	0.450	14.37	8.327
16	Quadratic	Reciprocal	4	656	7291	0.600	19.17	11.370
20	Quadratic	Reciprocal	6	2880	9569	0.616	19.79	13.810
24	Quadratic	Reciprocal	7	7040	11718	0.600	19.17	19.115
16	Linear	Reciprocal Root	8	6016	2337	0.450	14.37	5.145
16	Linear	LNS-Addition Log	7	2688	1999	0.450	14.38	4.515
16	Linear	Logarithm	8	6912	2425	0.450	14.38	5.817
16	Linear	Exponential	7	3840	3316	0.450	14.38	7.168
16	Quadratic	Reciprocal Root	4	1136	6047	0.600	19.16	9.366
16	Quadratic	LNS-Addition Log	5	960	4479	0.600	19.17	7.021
16	Quadratic	Logarithm	4	640	7135	0.600	19.17	10.803
16	Quadratic	Exponential	4	624	6850	0.600	19.17	10.448

Each of the designs in Table 7 are also synthesized using both constant-correction and variable-correction truncated-matrix multipliers and squarers. Table 8 gives results for 12-, 16- and 20-bit linear interpolators for the reciprocal function. The number of bits input to the interpolator, n, is given for each size. As discussed in Section 2.2, constants, such as the leading "1" for the reciprocal interpolator, are not input. The baseline designs using standard multipliers and squarers are listed as SMRTN under type. Constant-correction and variable-correction variants are listed as CCTM and VCTM, respectively. The number of multiplier columns that are formed, but rounded away in the output, k_{m1}, and the number of columns that are not formed in the truncated-matrix units, r_{m1}, are given. The minimum and maximum errors, E_{min} and E_{max}, as well as the variance of the error, E_{σ^2}, are given in ulps. The number of full adders (FAs) and half adders (HAs) used in the multiplier and the multi-operand adder are given for each interpolator. Area, delay and power estimates are given, normalized to the standard design.

Table 8. Linear interpolators, $f(x) = 1/x$. FA, full adder; HA, half adder; ulp, unit in the last place of the product.

12-bit input. $n = 11$, $m = 5$, $n_0 = 14$, $n_1 = 11$
Lookup Table Size $= 2^5(12 + 10) = 704$ bits

Type	k_{m1}	r_{m1}	E_{min} **(ulps)**	E_{max} **(ulps)**	E_{σ^2} **(ulps)**	**FAs**	**HAs**	**Area (Normalized)**	**Delay (Normalized)**	**Power (Normalized)**
SMRTN	6	0	−0.831	0.870	0.103	47	18	1.000	1.000	1.000
CCTM	1	5	−0.999	0.831	0.112	35	11	0.839	1.000	0.851
VCTM	2	4	−0.831	0.921	0.103	46	13	0.953	1.000	0.959

16-bit input. $n = 15$, $m = 8$, $n_0 = 18$, $n_1 = 10$
Lookup Table Size $= 2^8(16 + 9) = 6400$ bits

Type	k_{m1}	r_{m1}	E_{min} **(ulps)**	E_{max} **(ulps)**	E_{σ^2} **(ulps)**	**FAs**	**HAs**	**Area (Normalized)**	**Delay (Normalized)**	**Power (Normalized)**
SMRTN	7	0	−0.805	0.832	0.102	52	19	1.000	1.000	1.000
CCTM	2	5	−0.893	0.901	0.105	44	14	0.874	0.994	0.905
VCTM	1	6	−0.846	0.875	0.101	45	12	0.848	0.994	0.935

20-bit input. $n = 19$, $m = 10$, $n_0 = 22$, $n_1 = 12$
Lookup Table Size $= 2^{10}(20 + 11) = 31744$ bits

Type	k_{m1}	r_{m1}	E_{min} **(ulps)**	E_{max} **(ulps)**	E_{σ^2} **(ulps)**	**FAs**	**HAs**	**Area (Normalized)**	**Delay (Normalized)**	**Power (Normalized)**
SMRTN	9	0	−0.884	0.869	0.102	85	24	1.000	1.000	1.000
CCTM	3	6	−0.902	0.904	0.104	73	16	0.748	1.000	0.755
VCTM	1	8	−0.979	0.925	0.106	68	14	0.814	1.000	0.801

Parameters affecting the size of the lookup table are given for each design. This includes m, the number of address lines for the lookup table, as well as n_0 and n_1, the lengths of coefficients a_0 and a_1. The number of entries in the lookup table is 2^m. A careful examination of the coefficient values shows that some of the coefficient bits do not change and therefore do not need to be stored. For example, the first two bits in the 16-bit reciprocal lookup table are always "01" for a_0, and the first bit for a_1 is always "1", which can be hard wired rather than stored. Taking this into account, the size of the lookup table in bits is given for each interpolator size.

Table 9 shows results for 16-, 20- and 24-bit quadratic interpolators for the reciprocal function. As with Table 8, key parameters of each design are given, and the size of the lookup table is calculated. Additional parameters that are applicable only to quadratic interpolators are also given. This includes t, the number of LSBs in x_l not input to the squarer, and k and r values for the second multiplier and the squarer. As with the linear interpolators, error statistics, the number of FAs and HAs used in the multipliers, squarer and multi-operand adder and area, delay and power estimates are reported.

In order to demonstrate the generality of this method, interpolators for the remaining functions listed in Table 1 are designed and synthesized. Table 10 shows results for 16-bit linear interpolators, and Table 11 shows results for 16-bit quadratic interpolators. The reciprocal root interpolators require two lookup tables; one for the case where the exponent is even and the other for the case where it is odd.

Table 9. Quadratic interpolators, $f(x) = 1/x$.

16-bit input. $n = 15, m = 4, n_0 = 18, n_1 = 15, n_2 = 12, t_{sq} = 0$
Lookup Table Size $= 2^4(16 + 14 + 11) = 656$ bits

Type	k_{m1}	r_{m1}	k_{m2}	r_{m2}	k_{sq}	r_{sq}	E_{min} **(ulps)**	E_{max} **(ulps)**	E_{σ^2} **(ulps)**	**FAs**	**HAs**	**Area**	**Delay**	**Power**
												(Normalized)		
SMRTN	10	0	11	0	10	0	-0.925	0.993	0.105	303	43	1.000	1.000	1.000
CCTM	4	6	4	7	3	7	-0.905	0.993	0.106	264	33	0.855	1.000	0.863
VCTM	2	8	3	8	2	8	-0.963	0.993	0.105	263	28	0.865	1.000	0.908

20-bit input. $n = 19, m = 6, n_0 = 23, n_1 = 16, n_2 = 10, t_{sq} = 0$
Lookup Table Size $= 2^6(21 + 15 + 9) = 2880$ bits

Type	k_{m1}	r_{m1}	k_{m2}	r_{m2}	k_{sq}	r_{sq}	E_{min} **(ulps)**	E_{max} **(ulps)**	E_{σ^2} **(ulps)**	**FAs**	**HAs**	**Area**	**Delay**	**Power**
												(Normalized)		
SMRTN	12	0	11	0	14	0	-0.868	0.853	0.098	345	50	1.000	1.000	1.000
CCTM	2	10	3	8	2	12	-0.945	0.951	0.100	250	31	0.684	0.975	0.743
VCTM	2	10	2	9	1	13	-0.932	0.929	0.100	262	29	0.657	0.975	0.759

24-bit input. $n = 23, m = 7, n_0 = 26, n_1 = 20, n_2 = 13, t_{sq} = 1$
Lookup Table Size $= 2^7(24 + 19 + 12) = 7040$ bits

Type	k_{m1}	r_{m1}	k_{m2}	r_{m2}	k_{sq}	r_{sq}	E_{min} **(ulps)**	E_{max} **(ulps)**	E_{σ^2} **(ulps)**	**FAs**	**HAs**	**Area**	**Delay**	**Power**
												(Normalized)		
SMRTN	15	0	13	0	15	0	-0.963	0.934	0.099	539	68	1.000	1.000	1.000
CCTM	3	12	4	9	3	12	-0.922	0.982	0.099	417	43	0.842	1.000	0.867
VCTM	2	13	2	11	2	13	-0.940	0.992	0.099	414	35	0.836	1.000	0.847

Table 10. 16-bit linear interpolators, various functions.

Reciprocal Root. $n = 15, m = 7, n_0 = 18, n_1 = 9$
Lookup Table Size $= 2^7(16 + 8) + 2^7(15 + 8) = 6016$ bits

Type	k_{m1}	r_{m1}	E_{min} **(ulps)**	E_{max} **(ulps)**	E_{σ^2} **(ulps)**	**FAs**	**HAs**	**Area**	**Delay**	**Power**
								(Normalized)		
SMRTN	7	0	-0.937	0.968	0.118	55	18	1.000	1.000	1.000
CCTM	3	4	-0.937	0.959	0.118	50	15	1.001	1.000	0.984
VCTM	2	5	-0.965	0.962	0.116	51	12	0.830	1.000	0.931

LNS-Addition Logarithm. $K = 5, F = 11$. $n = 15, m = 7, n_0 = 14, n_1 = 8$
Lookup Table Size $= 2^7(14 + 7) = 2688$ bits

Type	k_{m1}	r_{m1}	E_{min} **(ulps)**	E_{max} **(ulps)**	E_{σ^2} **(ulps)**	**FAs**	**HAs**	**Area**	**Delay**	**Power**
								(Normalized)		
SMRTN	6	0	-0.970	0.917	0.092	48	13	1.000	1.000	1.000
CCTM	2	4	-0.927	0.971	0.093	43	10	0.873	1.000	0.944
VCTM	1	5	-0.927	0.990	0.094	44	8	0.852	1.000	0.924

Logarithm. $n = 15, m = 8, n_0 = 18, n_1 = 10$
Lookup Table Size $= 2^8(18 + 9) = 6912$ bits

Type	k_{m1}	r_{m1}	E_{min} **(ulps)**	E_{max} **(ulps)**	E_{σ^2} **(ulps)**	**FAs**	**HAs**	**Area**	**Delay**	**Power**
								(Normalized)		
SMRTN	6	0	-0.923	0.951	0.111	53	19	1.000	1.000	1.000
CCTM	2	4	-0.958	0.982	0.114	48	16	0.889	0.999	0.879
VCTM	1	5	-0.942	0.929	0.115	49	14	0.877	1.000	0.895

Table 10. *Cont.*

Exponential. $n = 16, m = 7, n_0 = 19, n_1 = 12$
Lookup Table Size = $2^7(19 + 11) = 3840$ bits

Type	k_{m1}	r_{m1}	E_{min} (ulps)	E_{max} (ulps)	E_{σ^2} (ulps)	FAs	HAs	Area	Delay	Power
								(Normalized)		
SMRTN	9	0	−0.935	0.929	0.113	85	20	1.000	1.000	1.000
CCTM	4	5	−0.890	0.977	0.111	78	12	0.879	1.000	0.893
VCTM	2	7	−0.964	0.950	0.112	74	10	0.827	1.000	0.899

Table 11. 16-bit quadratic interpolators, various functions.

Reciprocal Root. $n = 15, m = 4, n_0 = 18, n_1 = 13, n_2 = 9, t_{sq} = 0$
Lookup Table Size = $2^4(16 + 12 + 8) + 2^4(15 + 12 + 8) = 1136$ bits

Type	k_{m1}	r_{m1}	k_{m2}	r_{m2}	k_{sq}	r_{sq}	E_{min} (ulps)	E_{max} (ulps)	E_{σ^2} (ulps)	FAs	HAs	Area	Delay	Power
												(Normalized)		
SMRTN	10	0	10	0	11	0	−0.908	0.926	0.109	242	46	1.000	1.000	1.000
CCTM	2	8	1	9	1	10	−0.940	0.982	0.114	168	26	0.742	1.000	0.753
VCTM	1	9	0	10	1	10	−0.982	0.940	0.111	175	28	0.781	1.001	0.793

LNS-Addition Logarithm. $K = 5, F = 11$. $n = 15, m = 5, n_0 = 12, n_1 = 11, n_2 = 8, t_{sq} = 0$
Lookup Table Size = $2^5(11 + 11 + 8) = 960$ bits

Type	k_{m1}	r_{m1}	k_{m2}	r_{m2}	k_{sq}	r_{sq}	E_{min} (ulps)	E_{max} (ulps)	E_{σ^2} (ulps)	FAs	HAs	Area	Delay	Power
												(Normalized)		
SMRTN	10	0	10	0	10	0	−0.993	0.940	0.103	185	39	1.000	1.000	1.000
CCTM	4	6	5	5	4	6	−0.993	0.996	0.107	159	26	0.880	1.000	0.954
VCTM	2	8	3	7	2	8	−0.987	0.996	0.106	150	17	0.792	1.000	0.878

Logarithm. $n = 15, m = 4, n_0 = 18, n_1 = 15, n_2 = 10, t_{sq} = 0$
Lookup Table Size = $2^4(17 + 14 + 9) = 640$ bits

Type	k_{m1}	r_{m1}	k_{m2}	r_{m2}	k_{sq}	r_{sq}	E_{min} (ulps)	E_{max} (ulps)	E_{σ^2} (ulps)	FAs	HAs	Area	Delay	Power
												(Normalized)		
SMRTN	10	0	11	0	10	0	−0.947	0.852	0.107	281	48	1.000	1.000	1.000
CCTM	3	7	3	8	2	8	−0.864	0.971	0.108	226	32	0.778	1.000	0.869
VCTM	1	9	2	9	1	9	−0.912	0.925	0.111	226	23	0.777	1.000	0.896

Exponential. $n = 16, m = 4, n_0 = 19, n_1 = 15, n_2 = 9, t_{sq} = 0$
Lookup Table Size = $2^4(17 + 14 + 8) = 624$ bits

Type	k_{m1}	r_{m1}	k_{m2}	r_{m2}	k_{sq}	r_{sq}	E_{min} (ulps)	E_{max} (ulps)	E_{σ^2} (ulps)	FAs	HAs	Area	Delay	Power
												(Normalized)		
SMRTN	11	0	10	0	13	0	−0.908	0.849	0.107	287	47	1.000	1.000	1.000
CCTM	3	8	3	7	3	10	−0.855	0.991	0.110	225	32	0.821	1.000	0.891
VCTM	1	10	2	8	1	12	−0.941	0.951	0.109	217	35	0.827	1.000	0.834

From these results, it can be seen that a significant reduction in area and power can be achieved without exceeding the design specification of a ± 1 ulp error. For the linear-interpolator designs that are developed, area reductions up to 25.2% and power reductions up to 24.5% are obtained. For quadratic interpolators, area reductions up to 34.3% and power reductions up to 25.7% are obtained. If desired, savings in area can be traded-off for increased speed by changing the constraints in the synthesis tools.

There is no clear tendency for either constant correction or variable correction to produce better results. Constant-correction truncated-matrix multipliers and squarers with $r = r' - 1$ unformed columns have the same number of unformed partial products as variable-correction units with $r = r'$ unformed columns. For a given number of unformed partial products, constant-correction and variable-correction units have similar error characteristics. Due to the nonlinear nature of these units, however, the actual error bounds of the interpolator using them can only be determined through exhaustive simulation.

7. Future Work

The main contribution of this work is to extend [1] and show that standard multipliers and squarers can be replaced with truncated-matrix units in function interpolators to reduce area, delay and power for many different functions without changing the specified error bound.

A Chebyshev-series approximation is used to get initial coefficient values. Another approximation method, such as minimax, could be used. Minimax polynomials are known to be better approximations than Chebyshev polynomials. However, these are only initial values, which are modified using the algorithm described in Section 5.1. During the course of this work, it was found that the coefficients only change by a few ulps at most, so using a better initial approximation, such as minimax, would likely produce the same final results. The algorithm for finding the best coefficients is simple, and a better optimization method could be used. Since the coefficients only change by a few ulps, a better algorithm would again likely produce the same final results. While a better initial approximation method and a better optimization method may produce the same final results, they are worth investigating because they would probably reduce the run time of the overall method significantly, which would be beneficial when designing interpolators with larger operand sizes.

In this work, constant-correction [24] and variable-correction [27] methods are used to compensate for errors due to truncating partial-product bits in the multiplication and squaring matrices. These methods are simple to implement and have good error characteristics. Much work has been published in this area, with [32–34] being some of the more recent. Although these methods have better error characteristics, they are better by only a fraction of an ulp and require a more involved design procedure. Furthermore, the system-level correction provided by adjusting coefficients using the proposed algorithm may have a greater benefit to the simpler methods, because they have more room for improvement. However, it is worth investigating the use of fixed-width multipliers and squarers with better error characteristics, because they may further reduce area, delay and power and may do so more consistently as the operand size and the function being evaluated are changed.

Much work has been published on function approximation and evaluation, some of the more recent being [35–37]. It should be noted that the methods presented in this paper can be applied to most, if not all other designs for function approximation. This paper uses a piecewise polynomial evaluation based on a Chebyshev-series approximation as a starting point, because it is well known and produces good results. Designs for function evaluation that use other arithmetic structures, different methods for finding and optimizing coefficients or that are optimized for specific applications can also use the methods presented in this paper. Future work could be done to extend other designs by replacing standard arithmetic units with truncated-matrix versions and applying the methods in this work to adjust coefficients to maximize the number of partial-product bits that are eliminated.

8. Conclusions

Unlike many high-performance DSP kernels, function interpolators tend to require more precise results, so it would seem that they are not a good application for truncated-matrix units. However, this work shows that even applications with tight error specifications can benefit from using truncated-matrix multipliers and squarers. While this paper presents a technique for designing linear and quadratic interpolators, truncated-matrix units can be applied to most other design methods without changing the system architecture. The input and output datapaths of the original multipliers and squarers are not changed, making truncated-matrix units a "drop-in" replacement offering significant savings in area, delay and power.

Conflicts of Interest

The author declares no conflict of interest.

References

1. Walters, E.G., III; Schulte, M.J. Efficient Function Approximation Using Truncated Multipliers and Squarers. In Proceedings of the 17th IEEE Symposium on Computer Arithmetic, Cape Cod, MA, USA, 27–29 June 2005; pp. 232–239.
2. Tang, P.T.P. Table-Lookup Algorithms for Elementary Functions and Their Error Analysis. In Proceedings of the 10th IEEE Symposium on Computer Arithmetic, Grenoble, France, 26–28 June 1991; pp. 232–236.
3. Schulte, M.J.; Swartzlander, E.E., Jr. Exact Rounding of Certain Elementary Functions. In Proceedings of the 11th IEEE Symposium on Computer Arithmetic, Windsor, ON, Canada, 29 June–2 July 1993; pp. 138–145.
4. Schulte, M.J.; Swartzlander, E.E., Jr. Hardware Designs for Exactly Rounded Elementary Functions. *IEEE Trans. Comput.* **1994**, *43*, 964–973.
5. Takagi, N. Generating a Power of an Operand by a Table Look-Up and a Multiplication. In Proceedings of the 13th IEEE Symposium on Computer Arithmetic, Asilomar, CA, USA, 6–9 July 1997; pp. 126–131.
6. Das Sarma, D.; Matula, D.W. Faithful Interpolation in Reciprocal Tables. In Proceedings of the 13th IEEE Symposium on Computer Arithmetic, Asilomar, CA, USA, 6–9 July 1997; pp. 82–91.
7. Story, S.; Tang, P.T.P. New Algorithms for Improved Transcendental Functions on IA-64. In Proceedings of the 14th IEEE Symposium on Computer Arithmetic, Adelaide, SA, Australia, 14–16 April 1999; pp. 4–11.
8. Arnold, M.G.; Winkel, M.D. A Single-Multiplier Quadratic Interpolator for LNS Arithmetic. In Proceedings of the 2001 International Conference on Computer Design, Austin, TX, USA, 23–26 September 2001; pp. 178–183.
9. Arnold, M.G. Design of a Faithful LNS Interpolator. In Proceedings of the Euromicro Symposium on Digital System Design, Warsaw, Poland, 4–6 September 2001; pp. 336–344.
10. Pineiro, J.A.; Bruguera, J.D.; Muller, J.M. Faithful Powering Computation Using Table Look-Up and a Fused Accumulation Tree. In Proceedings of the 15th IEEE Symposium on Computer Arithmetic, Vail, CO, USA, 11–17 June 2001; pp. 40–47.
11. Cao, J.; Wei, B.W.Y.; Cheng, J. High-Performance Architectures for Elementary Function Generation. In Proceedings of the 15th IEEE Symposium on Computer Arithmetic, Vail, CO, USA, 11–17 June 2001; pp. 136–144.

12. Pineiro, J.A.; Bruguera, J.D. High-Speed Double-Precision Computation of Reciprocal, Division, Square Root, and Inverse Square Root. *IEEE Trans. Comput.* **2002**, *51*, 1377–1388.
13. Pineiro, J.A.; Oberman, S.F.; Muller, J.M.; Bruguera, J.D. High-Speed Function Approximation Using a Minimax Quadratic Interpolator. *IEEE Trans. Comput.* **2005**, *55*, 304–318.
14. Koren, I. *Computer Arithmetic and Algorithms*; Prentice Hall: Englewood Cliffs, NJ, USA, 1993.
15. Goldschmidt, R.E. Applications of Division by Convergence. Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1964.
16. Swartzlander, E.E., Jr.; Alexopoulos, A.G. The Sign/Logarithm Number System. *IEEE Trans. Comput.* **1975**, *C-24*, 1238–1242.
17. Vouzis, P.; Collange, S.; Arnold, M. LNS Subtraction Using Novel Cotransformation and/or Interpolation. In Proceedings of the 18th International Conference On Application-Specific Systems, Architectures, and Processors, Montreal, QC, Canada, 9–11 July 2007; pp. 107–114.
18. Coleman, J.N. Simplification of Table Structure in Logarithmic Arithmetic. *Electron. Lett.* **1995**, *31*, 1905–1906.
19. Arnold, M.G.; Bailey, T.; Cowles, J.; Winkel, M.D. Arithmetic Co-transformations in the Real and Complex Logarithmic Number Systems. *IEEE Trans. Comput.* **1998**, *47*, 777–786.
20. Arnold, M.G. Improved Cotransformation for LNS Subtraction. In Proceedings of the IEEE International Symposium on Circuits and Systems, Phoenix, AZ, USA, 26–29 May 2002; Volume 2, pp. 752–755.
21. Mathews, J.H. *Numerical Methods for Computer Science, Engineering, and Mathematics*; Prentice-Hall: Englewood Cliffs, NJ, USA, 1987.
22. Sadeghian, M.; Stine, J.E.; Walters, E.G., III. Optimized Cubic Chebyshev Interpolator for Elementary Function Hardware Implementations. In Proceedings of the IEEE International Symposium on Circuits and Systems, Melbourne, Australia, 1–5 June 2014; pp. 1536–1539.
23. Swartzlander, E.E., Jr. Truncated Multiplication with Approximate Rounding. In Proceedings of the 33rd Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 4–6 November 1999; Volume 2, pp. 1480–1483.
24. Schulte, M.J.; Swartzlander, E.E., Jr. Truncated Multiplication with Correction Constant. In *VLSI Signal Processing VI*; IEEE Press: Eindhoven, The Netherlands, 1993; pp. 388–396.
25. Wires, K.E.; Schulte, M.J.; Marquette, L.P.; Balzola, P.I. Combined Unsigned and Two's Complement Squarers. In Proceedings of the 33rd Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 4–6 November 1999; Volume 2, pp. 1215–1219.
26. Walters, E.G., III; Schulte, M.J.; Arnold, M.G. Truncated Squarers with Constant and Variable Correction. In Proceedings of the SPIE: Advanced Signal Processing Algorithms, Architectures, and Implementations XIV, Denver, CO, USA, 4–6 August 2004; Volume 5559, pp. 40–50.
27. King, E.J.; Swartzlander, E.E., Jr. Data-Dependent Truncation Scheme for Parallel Multipliers. In Proceedings of the 31st Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 2–5 November 1997; Volume 2, pp. 1178–1182.
28. Walters, E.G., III; Schulte, M.J. Fast, Bit-Accurate Simulation of Truncated-Matrix Multipliers and Squarers. In Proceedings of the 44th Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 7–10 November 2010; pp. 1139–1143.

29. Das Sarma, D.; Matula, D.W. Faithful Bipartite ROM Reciprocal Tables. In Proceedings of the 12th IEEE Symposium on Computer Arithmetic, Bath, UK, 19–21 July 1995; pp. 17–28.
30. Walters, E.G., III; Glossner, J.; Schulte, M.J. Automatic VHDL Model Generation of Parameterized FIR Filters. In Proceedings of the Second SAMOS Workshop on Embedded System Design Challenges, Systems, Architectures, Modeling, and Simulation, Samos, Greece, 22–25 July 2002.
31. Bickerstaff, K.; Schulte, M.J.; Swartzlander, E.E., Jr. Parallel Reduced Area Multipliers. *IEEE J. VLSI Signal Process.* **1995**, *9*, 181–191.
32. Shao, B.; Li, P. Array-Based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2015**, *62*, 1081–1090.
33. De Caro, D.; Petra, N.; Strollo, A.G.M.; Tessitore, F.; Napoli, E. Fixed-Width Multipliers and Multipliers-Accumulators With Min-Max Approximation Error. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2013**, *60*, 2375–2388.
34. He, W.Q.; Chen, Y.H.; Jou, S.J. High-Accuracy Fixed-Width Booth Multipliers Based on Probability and Simulation. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2015**, *62*, 2052–2061.
35. Lee, D.U.; Cheung, R.C.; Luk, W.; Villasenor, J.D. Hardware Implementation Trade-Offs of Polynomial Approximations and Interpolations. *IEEE Trans. Comput.* **2008**, *57*, 686–701.
36. Chen, P.Y.; Lien, C.Y.; Lu, C.P. VLSI Implementation of an Edge-Oriented Image Scaling Processor. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2009**, *17*, 1275–1284.
37. Chen, C.L.; Lai, C.H. Iterative Linear Interpolation Based on Fuzzy Gradient Model for Low-Cost VLSI Implementation. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2014**, *22*, 1526–1538.