PAPER

Multiplier-less and Table-less Linear Approximation for Square-Related Functions

In-Cheol PARK[†a)], *Member and* **Tae-Hwan KIM**[†b)], *Nonmember*

SUMMARY Square-related functions such as square, inverse square, square-root and inverse square-root operations are widely used in digital signal processing and digital communication algorithms, and their efficient realizations are commonly required to reduce the hardware complexity. In the implementation point of view, approximate realizations are often desired if they do not degrade performance significantly. In this paper, we propose new linear approximations for the square-related functions. The traditional linear approximations need multipliers to calculate slope offsets and tables to store initial offset values and slope values, whereas the proposed approximations exploit the inherent properties of square-related functions to linearly interpolate with only simple operations, such as shift, concatenation and addition, which are usually supported in modern VLSI systems. Regardless of the bit-width of the number system, more importantly, the maximum relative errors of the proposed approximations are bounded to 6.25% and 3.13% for square and square-root functions, respectively. For inverse square and inverse square-root functions, the maximum relative errors are bounded to 12.5% and 6.25% if the input operands are represented in 20 bits, respectively.

key words: *square, square-root, inverse square, inverse square-root, computer arithmetic, approximation, linear interpolation*

1. Introduction

Square and square-root operations are commonly used in many digital signal processing systems. For example, vector quantization determines the representative codeword by calculating the Euclidean distance [1] using square operations, Viterbi decoding computes branch metrics using square operations [2], and the maximum-likelihood (ML) estimation for Gaussian distortions, such as sphere detection for multi-input multi-output (MIMO) antenna systems [3], needs square operations. The square-root function is also important in many applications such as the distance calculation between two points in three-dimensional graphics and the calculation of Euclidean norm in vector median filtering [4].

In such applications, approximate square and square-root functions have more advantages than the exact ones. Especially for estimation algorithms dealing with input values corrupted by noise, exact calculations are less important than achieving efficient implementations. Therefore, it is desired to approximate the functions without sacrificing the error performance significantly. There have been a few works on the approximations. Some of the works are devoted to approximate square operations in the viewpoint of optimizing logic circuits [5]–[9], and others are focused on the algorithms to approximate Euclidean norm [10]–[12].

In this paper, we propose a new method to approximate square-related functions based on linear interpolation. This paper extends our previous work [13] for square and square-root by including the approximations of inverse square and inverse square-root. The proposed method exploits the following properties of the operations: 1) if the input value is a power of 2, its square and square-root can be calculated easily by shifting the input value, and 2) if the slope of every interpolating line is approximated as a power of 2, the slope multiplication can be replaced with a shift operation. Therefore, the proposed method enables both square and square-root functions to be approximated with simple operations such as shift, concatenation and addition. These operations are basically supported in most of modern VLSI systems, and can be shared for the proposed method. Note that the proposed method can be implemented without employing any tables and multipliers, whereas the traditional linear approximation necessitates them. The relative errors of the approximate square and square-root functions are bounded to 11.11% and 6.07%, respectively. Additionally, efficient compensation techniques are proposed to further reduce the maximum relative errors to 6.25% and 3.13%. The proposed multiplier-less, table-less linear approximation can be applied to approximate inverse square and inverse square-root functions, too.

The rest of the paper is organized as follows. In Sect. 2, we briefly describe previous works presented to approximate square and square-root functions. In Sect. 3, we explain the concept of the proposed multiplier-less, table-less linear approximations for the square-related functions. In Sects. 4 and 5, we present in detail how the proposed method can be applied to the square-related functions, and analyze their performances. In Sect. 6, we compare the performance of the proposed approximations with those of the previous works, and then discuss the advantages of the proposed method. Concluding remarks are made in Sect. 7.

2. Previous Works

In this section, we briefly review the previous works of square and square-root approximations. First of all, it is a well-known fact that the calculation of x^2 is simpler than the general multiplication because the number of partial prod-

Manuscript received April 2, 2010.
Manuscript revised June 7, 2010.

[†]The authors are with Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, 305–701, South Korea.
a) E-mail: icpark@ee.kaist.ac.kr
b) E-mail: bestcorean@gmail.com

DOI: 10.1587/transinf.E93.D.2979

ucts can be reduced by half [14]. This property has been actively utilized to approximate square functions in the estimation hardware such as Viterbi decoders. In [15], x^2 is exactly calculated by recursively decomposing x, and the decomposition results in a cellular logic array that can reduce hardware complexity compared to the general multiplier.

An N-bit positive number, x can be decomposed into $\{x_{N-1}, (N-1)b0\}$ and $x_{N-2:0}$ as in [15], where x_i is the i-th bit of x, $x_{i:j}$ is a part of x from the j-th bit to the i-th bit ($0 \leq j \leq i$), $\{x, y\}$ is the concatenation of x and y, and $Nb0$ is the N-bit binary representation of 0. Then x^2 can be expressed as $\{x_{N-1}, (N-1)b0\}^2 + 2\cdot\{x_{N-1}, (N-1)b0\}\cdot x_{N-2:0} + (x_{N-2:0})^2$. By removing non-dominant terms, we can approximate x^2 as $\{x_{N-1}, (N-1)b0\}\cdot(\{x_{N-1}, (N-1)b0\} + \{x_{N-2:0}, 1b0\})$ [8]. In order to improve the error performance of this approximation, a compensation scheme is introduced in [5], which adds a regular bit-pattern to the approximation result. The carry propagation mechanism occurring in summing the partial product terms of x^2 is investigated in [9] to derive a systematic approach to compensate the approximate error. In [7], the logic function for each bit in x^2 is approximated to a regular one and then followed by a heuristic compensation.

Although many applications require square-root operations, there have been few works on approximating the square-root operation. Possible solutions are to remove square-root operations by transforming the algorithm or to calculate them by using look-up tables. Compared with the square operation that can generate all the partial products simultaneously, the square-root operation is usually calculated iteratively as its computation is very similar to division [14]. To approximate the square-root operation, we can employ iterative solvers such as Newton-Rapshon, bisection and so on. As the iterative solvers need multiple cycles to produce the final result, they suffer from the converging speed, and are not appropriate for hardware implementation [16]. It is hard to find some previous works directly related to the approximation, since the square-root operation is serial in nature. Instead, we can find some works on the approximation of composite operations which contain square-root operations. In [11] and [12], the Euclidean norm (ℓ^2-norm) required in the vector median filtering is approximated as a linear combination of the components of a vector. In [10], the Euclidean norm is approximated as a linear combination of other norms such as ℓ^1- and ℓ^∞-norm. However, a number of complicated operations such as division and multiplication are used in those approximations [11], [12].

3. Multiplier-less and Table-less Linear Approximation

The value of a point between two known points can be linearly approximated by drawing a straight line between the two points. This method is widely used to piecewise approximate complicated functions. Figure 1 illustrates a general piecewise linear approximation, where the input range is partitioned into several segments and each segment is linearly approximated. To compute the output value for an input belonging to a segment, the *difference* between the input and the left end-point is multiplied by the *slope* and then it is added to the *initial offset* of the segment. Thus, piecewise linear approximation generally requires tables to store the *initial offset* and the *slope* for each segment and a multiplier to do the *slope* multiplication.

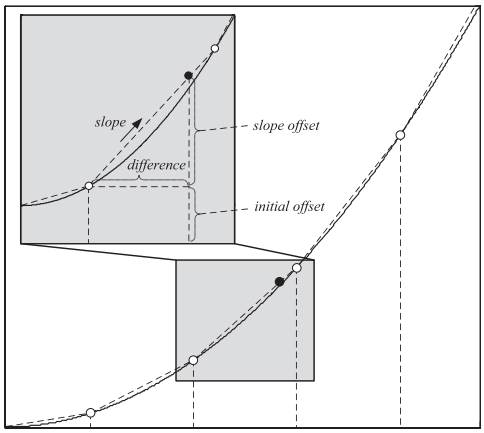

Fig. 1 General piecewise linear approximation.

As the piecewise linear approximation usually requires multipliers and tables, it seems to be inappropriate to be applied for the square-related functions. For example, the square function is exactly computed with one multiplication which is also required in a general linear approximation. Therefore, it seems to be meaningless to apply the linear approximation to the square function. If we can remove the table and the multiplication, however, the linear approximation can be a good candidate for efficient implementation.

The proposed method is based on the piecewise linear approximation, but it can be performed without any multiplications and tables. The fundamental concept behind the proposed method is to approximate square-related functions under two constraints. First, if the input is a power of 2, say 2^k, where k is a non-negative integer, we can calculate the square-related functions by shifting the input. For example, the square of 2^k can be calculated by shifting the input left by k bits. We can eliminate the offset table if the input range is segmented in such a way that the segment boundaries are at 2^k. Secondly, if the slope of a segment is restricted to a power of 2, we can also remove the multiplier, because the slope multiplication can be replaced with shifting the input difference. In Sects. 4 and 5, square-related functions such as square, square-root, inverse square and inverse square-root are approximated under the above constrains to achieve table-less and multiplier-less implementation.

4. Proposed Approximations for Square and Inverse Square

In this section, we propose new approximations for the square and the inverse square functions based on the multiplier-less and table-less interpolation technique. Simple error compensation techniques are also proposed based

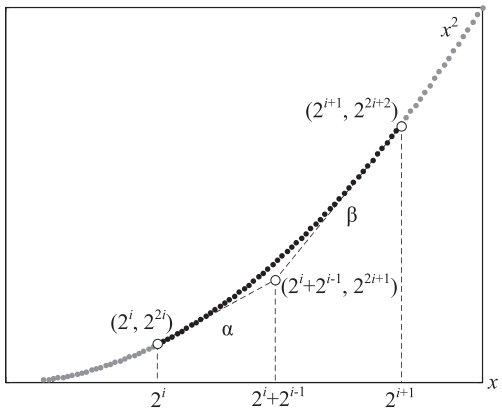

Fig. 2 Proposed piecewise linear approximation of x^2 for $2^i \le x < 2^{i+1}$.

on the analyses of the proposed approximations.

4.1 Proposed Linear Interpolation for Square

Let x be an N-bit positive number, where $N > 1$. The range of x is partitioned into N segments, each of which ranges from 2^i to 2^{i+1}, where i is an integer from 0 to $N-1$. At the two end-points of the i-th segment, the square values are 2^{2i} and 2^{2i+2} as shown in Fig. 2. If we use a single line for the approximation of the segment, its slope is not a power of 2, as $(2^{2i+2} - 2^{2i})/(2^{i+1} - 2^i) = 3 \cdot 2^i$. At the middle point in the segment, $(2^{i+1} + 2^i)/2 = 2^i + 2^{i-1}$, its square can be approximated as

$$(2^i + 2^{i-1})^2 = 2^{2i+1} + 2^{2i-2} \approx 2^{2i+1}. \tag{1}$$

With the approximation of (1), the segment whose input ranges from 2^i to 2^{i+1} can be interpolated with two lines denoted as α and β in Fig. 2. In this case, their slopes can be calculated as

$$\text{slope of } \alpha = \frac{2^{2i+1} - 2^{2i}}{2^i + 2^{i-1} - 2^i} = \frac{2^{2i}}{2^{i-1}} = 2^{i+1}, \tag{2}$$

and

$$\text{slope of } \beta = \frac{2^{2i+2} - 2^{2i+1}}{2^{i+1} - (2^i + 2^{i-1})} = \frac{2^{2i} \cdot 2}{2^i/2} = 2^{i+2}. \tag{3}$$

Note that both of the slopes are powers of 2. Therefore, the slope multiplication in the linear interpolation can be replaced with shifting, as multiplication by 2^i is the same as shifting left by i bits.

Let $f_1(x)$ be the square approximation based on the proposed linear interpolation. According to (2) and (3), $f_1(x)$ can be expressed with a number of line equations as follows:

$$f_1(x) = \begin{cases} 2^{2i} + (x - 2^i) \cdot 2^{i+1} & \text{for } 2^i \le x < 2^i + 2^{i-1} \\ 2^{2i+1} + \left(x - (2^i + 2^{i-1})\right) \cdot 2^{i+2} & \text{for } 2^i + 2^{i-1} \le x < 2^{i+1} \end{cases}. \tag{4}$$

Alternatively, this can be expressed as

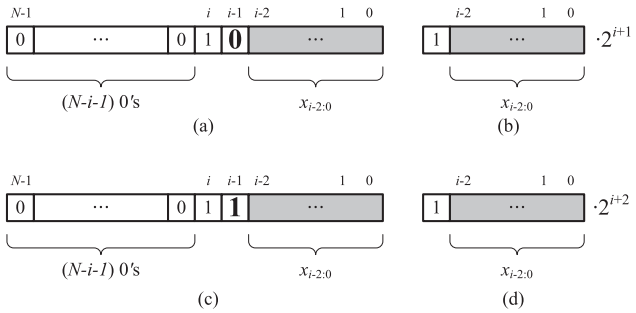

Fig. 3 Proposed method to calculate the approximate square. (a) $2^i \le x < 2^i + 2^{i-1}$, (b) approximate square of (a), (c) $2^i + 2^{i-1} \le x < 2^{i+1}$, and (d) approximate square of (c).

$$f_1(x) = \begin{cases} \left(2^{i-1} + (x - 2^i)\right) \cdot 2^{i+1} & \text{for } 2^i \le x < 2^i + 2^{i-1} \\ \left(2^{i-1} + \left(x - (2^i + 2^{i-1})\right)\right) \cdot 2^{i+2} & \text{for } 2^i + 2^{i-1} \le x < 2^{i+1} \end{cases}. \tag{5}$$

The additions in (5), $2^{i-1} + (x - 2^i)$ and $2^{i-1} + (x - (2^i + 2^{i-1}))$, cause no carry propagation and thus can be replaced with simple concatenations. Considering the range of x specified in the first equation of (5), $x_{i:i-1}$ is 2b10, where 2b10 means the 2-bit binary number 10. This means that $x - 2^i$ can be expressed as $x_{i-2:0}$ as shown in Fig. 3 (a). It is clear that adding 2^{i-1} to $x_{i-2:0}$ does not cause any carry propagation. Similarly, $x-(2^i+2^{i-1})$ in the second equation of (5) is $x_{i-2:0}$, because $x_{i:i-1}$ = 2b11 in the input range as shown in Fig. 3 (c). Hence, the addition contained in the second equation of (5), $2^{i-1} + x_{i-2:0}$, does not cause carry propagation, either. With these properties, the above equations can be simplified with adopting concatenations as follows:

$$f_1(x) = \begin{cases} \{1, x_{i-2:0}\} \cdot 2^{i+1} & \text{for } 2^i \le x < 2^i + 2^{i-1} \\ \{1, x_{i-2:0}\} \cdot 2^{i+2} & \text{for } 2^i + 2^{i-1} \le x < 2^{i+1} \end{cases}. \tag{6}$$

Suppose that the left most non-zero position in x is i. The proposed approximate square can be made by concatenating 1 in front of $x_{i-2:0}$ and then shifting it left by the left most non-zero position plus a constant. The constant is determined by x_{i-1}. More specifically, the constant is 1 when x_{i-1} is 0, or 2 when x_{i-1} is 1. Therefore, the proposed square approximation can be simplified as follows:

$$f_1(x) = \begin{cases} \{1, x_{i-2:0}\} \cdot 2^{i+1} & \text{for } x_{i-1} = 0 \\ \{1, x_{i-2:0}\} \cdot 2^{i+2} & \text{for } x_{i-1} = 1 \end{cases}. \tag{7}$$

Figures 3 (b) and (d) are the proposed approximation of the number presented in Figs. 3 (a) and (c), respectively.

4.2 Error Analysis and Error Compensation of the Approximate Square

As the proposed square approximation has errors compared to the exact square values, we analyze the relative error defined below,

$$RE(f_1(x)) = \left| f_1(x) - x^2 \right| / x^2. \tag{8}$$

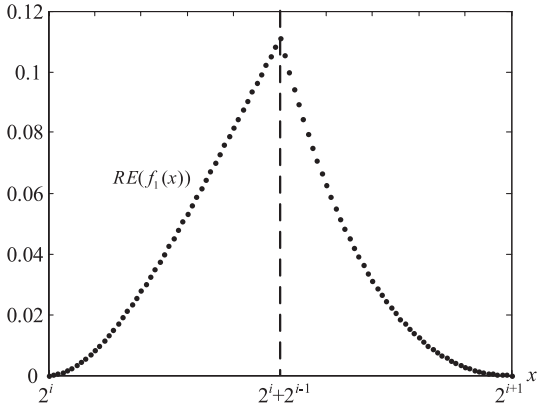

Fig. 4 Relative error of the proposed approximate square for $2^i \le x < 2^{i+1}$.

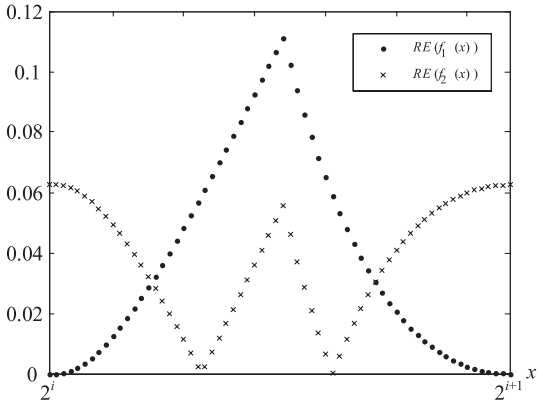

Fig. 5 Relative error of the proposed approximate square with the error compensation for $2^i \le x < 2^{i+1}$.

Table 1 Relative error of the proposed approximate square.

Bit-width of x	$MAX(RE(f_1(x)))$	$AVG(RE(f_1(x)))$
4	0.1111	0.0352
8	0.1111	0.0375
12	0.1111	0.0382
16	0.1111	0.0383
20	0.1111	0.0383

Substituting (4) into (8), we obtain

$$RE(f_1(x)) = \begin{cases} \dfrac{\left|-x^2+x\cdot 2^{i+1}+2^{2i}-2^{2i+1}\right|}{x^2} & \text{for} \quad 2^i \le x < 2^i+2^{i-1} \\ \dfrac{\left|-x^2+x\cdot 2^{i+2}-2^{2i+2}\right|}{x^2} & \text{for} \quad 2^i+2^{i-1} \le x < 2^{i+1} \end{cases}. \tag{9}$$

Figure 4 shows the plot of $RE(f_1(x))$ for $2^i \le x < 2^{i+1}$. $RE(f_1(x))$ is maximized when x is equal to $2^i + 2^{i-1}$, which is calculated as

$$MAX(RE(f_1(x))) = \frac{\left|f_1(2^i+2^{i-1})-(2^i+2^{i-1})^2\right|}{(2^i+2^{i-1})^2} = \frac{1}{9} \approx 0.1111. \tag{10}$$

Additionally, average relative error of $f_1(x)$ is defined as

$$AVG(RE(f_1(x))) = \left(\sum_{x=1}^{2^N-1} RE(f_1(x))\right) / \left(2^N-1\right). \tag{11}$$

The average relative errors in (11) are evaluated by a computer program. In Table 1, these are listed together with the maximum relative errors for various bit-widths of x. Note that the maximum relative error of the proposed approximation is constant regardless of the bit-widths, and the average relative error is almost constant. The relative error constancy is due to the inherent property of the proposed linear approximation, which guarantees good performance even for large numbers.

The proposed square approximation, $f_1(x)$, is always smaller than or equal to the exact value of x^2, and its relative error can be 11.11% maximally. Therefore, we can scale up $f_1(x)$ by a factor of 17/16 when the smaller relative error is required in some applications. This scaling can be achieved by simply adding $f_1(x)\cdot 2^{-4}$ to $f_1(x)$. Let $f_2(x)$ be the error-compensated approximation of x^2; that is, $f_2(x) = f_1(x) + (f_1(x)\cdot 2^{-4})$. Figure 5 illustrates the relative errors resulting from $f_2(x)$ and $f_1(x)$ for the i-th segment, where we can see that the maximum relative error of $f_2(x)$ is reduced to almost a half compared to that of $f_1(x)$, but their average errors are almost equal to each other. In Table 2, the maximum and average relative errors of $f_2(x)$ are listed for various bit-widths.

4.3 Proposed Linear Interpolation for Inverse Square

We can apply the proposed multiplier-less, table-less linear interpolation to inverse square function. Similar to the approximation proposed for square function, $1/x^2$ can be approximated by piecewise linear interpolations as illustrated in Fig. 6. We assume that x is greater than zero, as we are dealing with the reciprocal function in this sub-section. Simple operations such as shifting and concatenation are also enough in the proposed linear interpolation. For example, if $x = 2^k$, where k is an integer not less than 0, $1/x^2$ can be approximated by shifting x right by $3k$. In addition, the slopes of the lines employed in the linear interpolations are also powers of 2, making it possible to replace the slope multiplications with shift operations.

Let $f_3(x)$ be the proposed approximation of $1/x^2$, where x is an N-bit positive number. This approximation can be expressed with a number of line equations as follows,

$$f_3(x) = \begin{cases} 2^{-2i}-(x-2^i)\cdot 2^{-3i} & \text{for } 2^i \le x < 2^i+2^{i-1} \\ 2^{-2i-1}-(x-2^i-2^{i-1})\cdot 2^{-3i-1} & \text{for } 2^i+2^{i-1} \le x < 2^{i+1} \end{cases}, \tag{12}$$

where i is an integer ranging from 0 to $N-1$. In the 2's complement representation, the difference between the negated

Table 2 Relative error of the proposed approximate square with error compensation.

Bit-width of x	$MAX(RE(f_2(x)))$	$AVG(RE(f_2(x)))$	$\frac{MAX(RE(f_2(x)))}{MAX(RE(f_1(x)))}$	$\frac{AVG(RE(f_2(x)))}{AVG(RE(f_1(x)))}$
4	0.0625	0.0466	0.5625	1.3238
8	0.0625	0.0383	0.5625	1.0213
12	0.0625	0.0373	0.5625	0.9764
16	0.0625	0.0372	0.5625	0.9712
20	0.0625	0.0372	0.5625	0.9712

Table 3 Relative error of the proposed approximate inverse square.

Bit-width of x	$MAX(RE(f_3(x)))$	$AVG(RE(f_3(x)))$	$MAX(RE(f_4(x)))$	$AVG(RE(f_4(x)))$
4	0.1816	0.0917	0.1250	0.0572
8	0.1852	0.1232	0.1250	0.0346
12	0.1852	0.1272	0.1250	0.0317
16	0.1852	0.1278	0.1250	0.0314
20	0.1852	0.1276	0.1250	0.0314

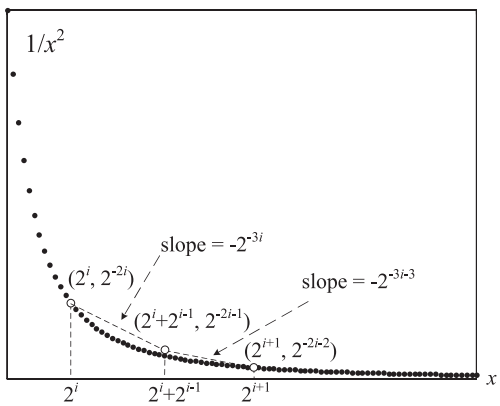

Fig. 6 Proposed piecewise linear approximation of $1/x^2$ for $2^i \le x < 2^{i+1}$.

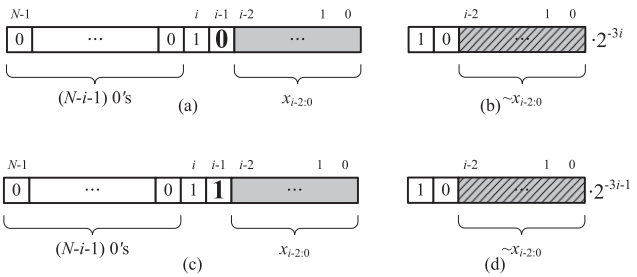

Fig. 7 Proposed method to calculate the approximate inverse square. (a) $2^i \le x < 2^i + 2^{i-1}$, (b) approximate inverse square of (a), (c) $2^i + 2^{i-1} \le x < 2^{i+1}$, and (d) approximate inverse square of (c).

one and the inverted one is only one. We adopt the inversion to replace the subtraction in (12) with addition, and then the addition can be achieved by a simple concatenation as in the case of the proposed square approximation. Moreover, the range of x in (12) can be determined by examining the left-most non-zero bit-patterns. Suppose that the most significant $N-i-1$ bits are zeros in the binary representation of x. In this case, the proposed approximation of $1/x^2$ can be expressed as

$$f_3(x) \approx \begin{cases} \{2b10, \sim x_{i-2:0}\}\cdot 2^{-3i} & \text{for } x_{i-1} = 0 \\ \{2b10, \sim x_{i-2:0}\}\cdot 2^{-3i-1} & \text{for } x_{i-1} = 1 \end{cases}, \quad (13)$$

where $\sim x$ means the inversion of x. Figure 7 illustrates the proposed approximation of inverse square corresponding to (13). In the proposed approximation, we detect the leading non-zero bit-pattern of x, invert its sub-bits, and shift it after attaching a prefix determined by the leading non-zero bit-pattern.

For various bit-widths, the relative errors of $f_3(x)$ are listed in Table 3. Different from the square approximations, the relative error is not maximized at the middle point of a segment. As the maximum relative error is dependent on the segment location, it is not constant but close to a constant value; about 18.5%. Like the previous sub-section, we can apply simple compensation techniques to reduce the relative error. Let $f_4(x) = f_3(x) - (f_3(x)\cdot 2^{-3})$ be the error-compensated approximations for $1/x^2$. As shown in Table 3, the suggested error compensations improve the maximum relative errors to 12.5%.

5. Proposed Approximations for Square-Root and Inverse Square-Root

In this section, we propose new approximations for the square-root and the inverse square-root functions, and analyze their error performances. The proposed methods are also based on piecewise linear approximations, and do not need any multiplications and tables.

5.1 Proposed Linear Interpolation for Square-Root

Let x be a $2N$-bit positive number, where $N > 1$. The entire range of x is partitioned into N segments, each of which ranges from 2^{2i} to 2^{2i+2}, where i is an integer from 0 to $N-1$. The i-th segment is shown in Fig. 8. At the middle point in this segment, 2^{2i+1}, its square-root can be approximated as

$$2^{(2i+1)/2} = \sqrt{2}\cdot 2^i \approx (1 + 1/2)\cdot 2^i = 2^i + 2^{i-1}. \quad (14)$$

By taking (14), the curve of $\sqrt{x}$ in the i-th segment can be linearly approximated with two lines, γ and δ, as depicted in Fig. 8. Let $f_5(x)$ be the proposed linear interpolation of

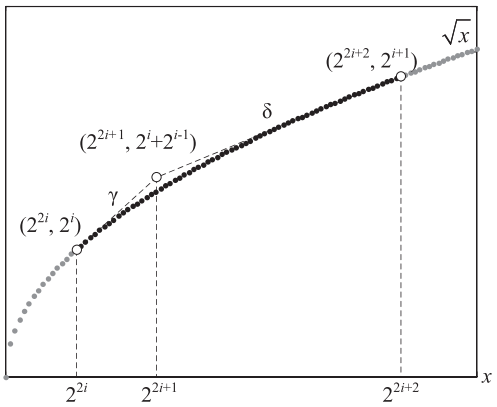

Fig. 8 Proposed piecewise linear approximation of $\sqrt{x}$ for $2^{2i} \le x < 2^{2i+2}$.

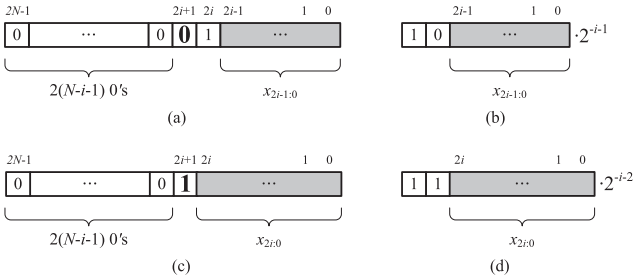

Fig. 9 Proposed method to calculate the approximate square-root. (a) $2^{2i} \le x < 2^{2i+1}$, (b) approximate square-root of (a), (c) $2^{2i+1} \le x < 2^{2i+2}$, and (d) approximate square root of (c).

the square-root of x. Then, $f_5(x)$ can expressed with a number of line equations as follows:

$$f_5(x) = \begin{cases} 2^i + (x - 2^{2i}) \cdot 2^{-i-1} \\ = \left(2^{2i+1} + (x - 2^{2i})\right) \cdot 2^{-i-1} & \text{for } 2^{2i} \le x < 2^{2i+1} \\ 2^i + 2^{i-1} + (x - 2^{2i+1}) \cdot 2^{-i-2} \\ = \left(2^{2i+2} + 2^{2i+1} + (x - 2^{2i+1})\right) \cdot 2^{-i-2} & \text{for } 2^{2i+1} \le x < 2^{2i+2} \end{cases} \tag{15}$$

If x is less than 2^{2i+1}, $x - 2^{2i}$ in (15) can be replaced with $x_{2i-1:0}$, and if x is less than 2^{2i+2}, $x - 2^{2i+1}$ in (15) can be replaced with $x_{2i:0}$. Figures 9 (a) and (c) correspond to these cases, respectively. In (15), $2^{2i+1} + x_{2i-1:0}$ and $2^{2i+2} + 2^{2i+1} + x_{2i:0}$ can be achieved by doing simple concatenations, $\{2b10, x_{2i-1:0}\}$ and $\{2b11, x_{2i:0}\}$, respectively. For a given x, the binary bit-pattern of x is divided into groups of two adjacent bits starting from the least significant bit, and then we search for the left-most non-zero group to find the range of x. If the group is $x_{2i+1:2i}$, $f_5(x)$ can be expressed as follows:

$$f_5(x) = \begin{cases} \{2b10, x_{2i-1:0}\} \cdot 2^{-i-1} & \text{for } x_{2i+1} = 0 \\ \{2b11, x_{2i:0}\} \cdot 2^{-i-2} & \text{for } x_{2i+1} = 1 \end{cases}. \tag{16}$$

The proposed square-root approximations corresponding to Figs. 9 (a) and (c) are shown in Figs. 9 (b) and (d), respectively.

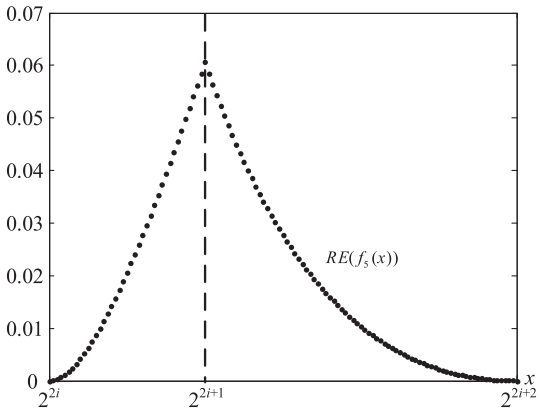

Fig. 10 Relative error of the proposed approximate square-root for $2^{2i} \le x < 2^{2i+2}$.

5.2 Error Analysis and Error Compensation of the Approximate Square-Root

The relative error of $f_5(x)$ is defined below:

$$RE(f_5(x)) = \left|f_5(x) - \sqrt{x}\right| / \sqrt{x}. \tag{17}$$

Substituting (15) into (17), we obtain

$$RE(f_5(x)) = \begin{cases} \dfrac{\left|x \cdot 2^{-i-1} - \sqrt{x} - 2^{-i-1} + 2^i\right|}{\sqrt{x}} & \text{for } 2^{2i} \le x < 2^{2i+1} \\ \dfrac{\left|x \cdot 2^{-i-2} - \sqrt{x} + 2^i\right|}{\sqrt{x}} & \text{for } 2^{2i+1} \le x < 2^{2i+2} \end{cases}, \tag{18}$$

where x is in the i-th segment. Figure 10 shows the plot of (18), the relative error is maximized when x is 2^{2i+1}, which is calculated as

$$\begin{aligned} MAX(RE(f_5(x)) &= \frac{\left|f_5(x) - \sqrt{(2^{2i+1})}\right|}{\sqrt{2^{2i+1}}} \\ &= \frac{3/2 - \sqrt{2}}{\sqrt{2}} \approx 0.0607. \end{aligned} \tag{19}$$

Similar to (11), the average relative error of $f_5(x)$ is defined as

$$AVG(RE(f_5(x))) = \left(\sum_{x=1}^{2^{2N}-1} RE(f_5(x))\right) / \left(2^{2N} - 1\right). \tag{20}$$

The average relative errors evaluated for various bit-widths are listed in Table 4 along with the maximum relative errors. Note that the maximum relative error is constant regardless of the bit-widths of the number systems, like the proposed approximate square described in the previous section.

$f_5(x)$ is always larger than or equal to the exact square-root value and its relative error is up to 6.07% as shown in Table 4. To reduce the maximum error, we can apply a

Table 5 Relative error of the proposed approximate square-root with error compensation.

Bit-width of x	$MAX(RE(f_6(x)))$	$AVG(RE(f_6(x)))$	$\frac{MAX(RE(f_6(x)))}{MAX(RE(f_5(x)))}$	$\frac{AVG(RE(f_6(x)))}{AVG(RE(f_5(x)))}$
4	0.0313	0.0197	0.5157	1.0314
8	0.0313	0.0193	0.5157	1.0105
12	0.0313	0.0193	0.5157	1.0105
16	0.0313	0.0193	0.5157	1.0105
20	0.0313	0.0193	0.5157	1.0105

Table 4 Relative error of the proposed approximate square-root.

Bit-width of x	$MAX(RE(f_5(x)))$	$AVG(RE(f_5(x)))$
4	0.0607	0.0191
8	0.0607	0.0191
12	0.0607	0.0191
16	0.0607	0.0191
20	0.0607	0.0191

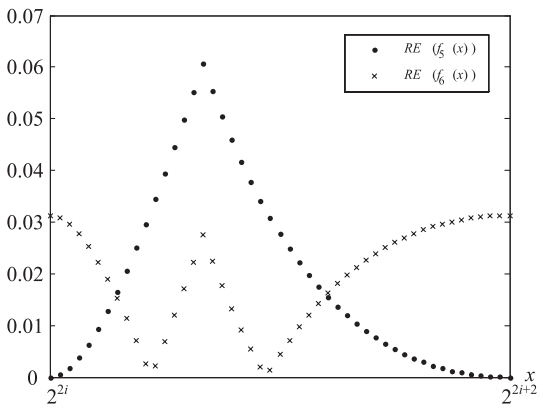

Fig. 11 Relative error of the proposed approximate square-root with the error compensation for $2^{2i} \le x < 2^{2i+2}$.

compensation technique similar to that used in the previous section. As $f_5(x)$ is always larger than $\sqrt{x}$, we can scale down $f_5(x)$ by a factor of 31/32 to achieve the smaller maximum relative error. Let $f_6(x)$ be the compensated one; that is, $f_6(x) = f_5(x) - (f_5(x) \cdot 2^{-5})$. Figure 11 compares the error performances of $f_5(x)$ and $f_6(x)$. Applying the proposed compensation reduces the maximum relative error to almost a half, while maintaining the average relative error performance. In Table 5, the maximum and average relative errors of $f_6(x)$ are listed for various bit-widths.

5.3 Proposed Linear Interpolation for Inverse Square-Root

A similar method can be applied to approximate inverse square-root. As shown in Fig. 12, the proposed approximation of the inverse square-root of a $2N$-bit positive number x, $f_7(x)$, can be expressed with a number of line equations as follows:

$$f_7(x) = \begin{cases} 2^{-i} - (x - 2^{2i}) \cdot 2^{-3i-2} & \text{for } 2^{2i} \le x < 2^{2i+1} \\ 2^{-i-1} + 2^{-i-2} - (x - 2^{2i+1}) \cdot 2^{-3i-3} & \text{for } 2^{2i+1} \le x < 2^{2i+2} \end{cases}, \tag{21}$$

where i is an integer ranging from 0 to $N - 1$. For a given

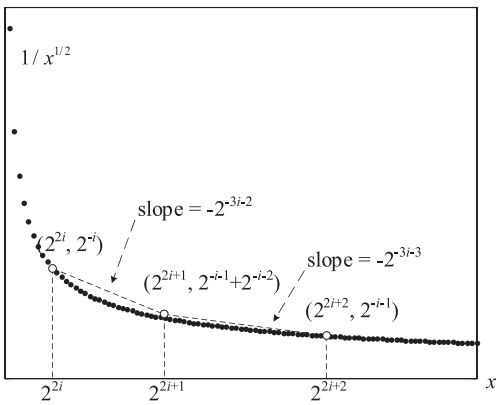

Fig. 12 Proposed piecewise linear approximation of $1/\sqrt{x}$ for $2^{2i} \le x < 2^{2i+2}$.

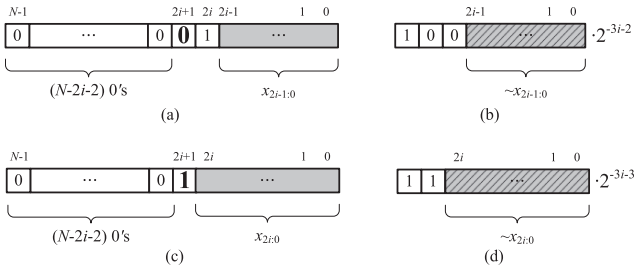

Fig. 13 Proposed method to calculate the approximate inverse square-root. (a) $2^{2i} \le x < 2^{2i+1}$, (b) approximate inverse square of (a), (c) $2^{2i+1} \le x < 2^{2i+2}$, and (d) approximate inverse square of (c).

x, the binary bit-pattern of x is divided into groups of two adjacent bits starting from the least significant bit. The leftmost non-zero group is searched to find the range of x. If the group is $x_{2i+1:2i}$, $f_7(x)$ can be expressed as

$$f_7(x) \approx \begin{cases} \{3b100, \sim x_{2i-1:0}\} \cdot 2^{-3i-2} & \text{for } x_{2i+1} = 0 \\ \{2b11, \sim x_{2i:0}\} \cdot 2^{-3i-3} & \text{for } x_{2i+1} = 1 \end{cases}. \tag{22}$$

Figure 13 illustrate the proposed approximation of inverse square-root corresponding to (22). For various bit-widths, the relative errors of $f_7(x)$ are listed in Table 6. A simple compensation technique can also be applied to reduce the relative errors. Let $f_8(x) = f_7(x) - (f_7(x) \cdot 2^{-4})$ be the error-compensated approximation for $1/\sqrt{x}$. As shown in Table 6, the maximum relative errors can be improved to 6.25% when the suggested error compensation is used.

Table 6 Relative error of the proposed approximate inverse square-root.

Bit-width of x	$MAX(RE(f_7(x)))$	$AVG(RE(f_7(x)))$	$MAX(RE(f_8(x)))$	$AVG(RE(f_8(x)))$
4	0.0887	0.0571	0.0625	0.0214
8	0.0887	0.0621	0.0625	0.0174
12	0.0887	0.0628	0.0625	0.0169
16	0.0887	0.0628	0.0625	0.0168
20	0.0887	0.0629	0.0625	0.0168

6. Comparison and Discussion

In Figs. 14 and 15, we compare the relative error performance of the proposed square approximation with those of previous works. In fact, the error performances in large number systems are much more important than those in small number systems, as the exact square calculation can be realized with a few logic gates when the number system is represented in a small number of bits. We can see in the figures that the proposed method is associated with constant error performance, while those of the previous works are severely degraded as the bit-width increases. In the proposed square-root approximation, the relative errors of the compensated one are always less than 3.13%. In contrast to the iterative approximations, the proposed methods have no convergence problems, as they can be computed at once. In addition, it is worth noting that the proposed approximations maintain the monotonic behaviors; that is, $f_1(a) \geq f_1(b)$, $f_3(a) \leq f_3(b)$, $f_5(a) \geq f_5(b)$ and $f_7(a) \leq f_7(b)$ if $a \leq b$. These monotonic properties are significant in comparing two approximated values.

The proposed approximations for square and square-root functions can be understood by employing the 1st-order Taylor series expansion under the first constraint that the segment boundaries are at 2^i. For example, the proposed approximate square in (4) is the same as the composition of the 1st-order Taylor expansions for $2^i \leq x < 2^i + 2^{i-1}$ and $2^i + 2^{i-1} \leq x < 2^{i+1}$. This is a coincidence, since the proposed method is not derived from the Taylor series expansion. In addition, the proposed method has a contribution in selecting specific evaluation points of the Taylor expansion that can result in multiplier-less and table-less implementations. Moreover, simple calculation methods are derived as expressed in (7) and (16). Contrary to the cases of square and square-root functions, the proposed approximations for inverse square and inverse square-root functions are quite different from those based on the Taylor expansion, because the slope of the proposed interpolation is different from that of the tangent line resulting from the Taylor expansion. As exemplified in Fig. 16, both the proposed method and the Taylor expansion consist of line segments whose slopes are powers of 2 for the inverse square function, but their slopes are different to each other. If the slopes resulting from the 1st order Taylor expansion are used, the middle point where two adjacent slopes are met has an x-coordinate of $2^i + 2^{i+1}/7$. As the x-coordinate has a long series of 1 s and 0 s in the binary representation, we have to compare the given input to the x-coordinate of the middle point to decide which line segment should be used. Therefore, employing the Taylor expansion under the first constraint leads to a more complicated implementation than the proposed method.

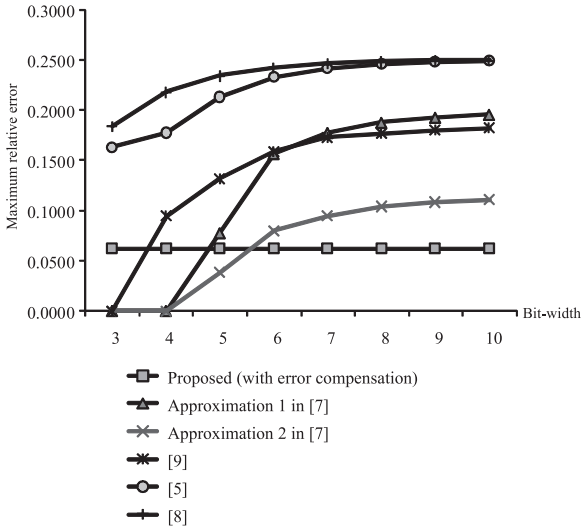

Fig. 14 Comparison of the maximum relative errors of square approximations.

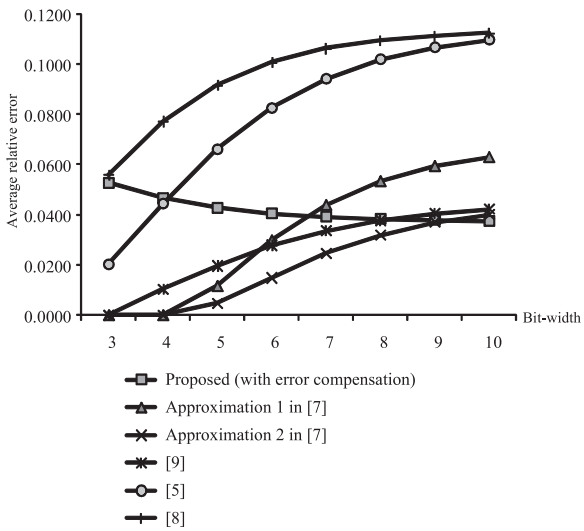

Fig. 15 Comparison of the average relative errors of square approximations.

The proposed method produces approximation results of square-related functions in a non-iterative way, which is invaluable in particular for the computations of square-root, inverse square, and inverse square functions. Since

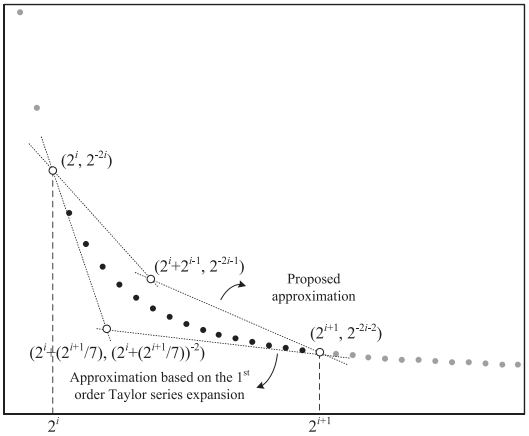

Fig. 16 Comparison between the proposed inverse square approximation one and that based on the 1st order Taylor series expansion.

Table 7 Comparison of 32-bit square-root implementations.

	Exact	Proposed
Gate count	4152.4	1126.4
Critical path length	9.95 ns	1.87 ns
Max. relative error	0	3.13%

the square-root and inversion are usually performed in iterative ways [14], their circuit implementations are associated with long delay if implemented in combinational circuits. To evaluate the proposed method in the perspective of real circuits, a 32-bit square-root function was implemented and compared with the exact square-root unit provided in Synopsys DesignWare [17]. Both the exact and the proposed square functions were described in Verilog HDL, and synthesized using Synopsys Design Compiler [18] with a 0.18 μm CMOS cell library. The gate count and the critical path delay estimated with the logic synthesis results are summarized in Table 7. In the table, the gate counts are estimated by counting the smallest two-input NAND as one. As shown in Table 7, the proposed method requires much less logic gates, and its critical path is very short. These merits will be more prominent when the bit-width becomes larger.

The proposed multiplier-less, table-less linear interpolation requires simple basic operations, such as shift, concatenation and detection of the leading non-zero bit, which are usually supported in modern VLSI systems. To compensate the relative errors, an addition is needed additionally. Therefore, it is possible to realize the proposed method by sharing the basic operators provided in the VLSI systems. For example, instructions for shifting and counting leading zeros are supported in most of contemporary processor architectures, as specified in [19], [20]. Hence, the proposed approximations can be efficiently implemented with the assistance of those pre-defined instructions.

7. Conclusion

Square-related functions are fundamental operations that are widely used in many digital signal processing systems. In many cases, it is desired to efficiently approximate them without degrading the performance severely, rather than to calculate them exactly. In this paper, we have proposed new methods to approximate square-related functions in a fixed point number system. Though the proposed method is based on the piecewise linear approximation, it can be implemented without employing tables and multipliers. Simple operations such as shift, concatenation, and addition, which are commonly supported in VLSI systems, are only used in the proposed approximation. For the proposed approximate square and square-root functions, the mathematical analysis reveals that the proposed approximations have no convergence problems, the maximum relative errors are constant irrespective of bit-widths, and the average relative errors are also almost constant. In addition, simple compensation techniques are proposed to further reduce the maximum relative errors. If the simple compensation techniques are employed, the maximum relative errors are 6.25% and 3.13% for the approximate square and square-root functions, respectively. The proposed method can be applied to inverse square and inverse square-root functions, and for 20-bit numbers their maximum relative errors are up to 12.5% and 6.25%, respectively.

References

[1] M.R. Soleymani and S.D. Morgera, "A fast MMSE encoding technique for vector quantization," IEEE Trans. Commun., vol.37, no 6, pp.656–659, June 1989.
[2] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," IEEE Trans. Inf. Theory, vol.IT-13, no.2, pp.260–269, April 1967.
[3] B.M. Hochwald and S.T. Brink, "Achieving near-capacity on a multiple-antenna channel," IEEE Trans. Commun., vol.51, no.3, pp.389–399, March 2003.
[4] J. Astola, P. Haavisto, and Y. Neuvo, "Vector median filter," Proc. IEEE, vol.78, no.4, pp.690–710, April 1990.
[5] A.A. Hiasat and H.S. Abdel-Aty-Zohdy, "Combinational logic approach for implementing an improved approximate squaring function," IEEE J. Solid-State Circuits, vol.34, no.2, pp.236–240, Feb. 1999.
[6] J.-T. Yoo, K.F. Smith, and G. Gopalakrishnan, "A fast parallel squarer based on divide-and-conquer," IEEE J. Solid-State Circuits, vol.32, no.6, pp.909–912, June 1997.
[7] J.M.P. Langlois and D. Al-Khalili, "Carry-free approximate squaring functions with $O(n)$ complexity and $O(1)$ delay," IEEE Trans. Circuits Syst. II, Express Briefs, vol.53, no.5, pp.374–378, May 2006.
[8] A. Eshraghi, T.S. Fiez, K.D. Winters, and T.R. Fischer, "Design of a new squaring function for the Viterbi algorithm," IEEE J. Solid-State Circuits, vol.29, no.9, pp.1102–1107, Sept. 1994.
[9] M.-H. Sheu and S.-H. Lin, "Fast compensative design approach for the approximate squaring function," IEEE J. Solid-State Circuits, vol.37, no.1, pp.95–97, Jan. 2002.
[10] C. Seol and K. Cheun, "A low complexity Euclidean norm approximation," IEEE Trans. Signal Process., vol.56, no.4, pp.1721–1726, April 2008.
[11] M. Barni, F. Buti, F. Bartolini, and V. Cappellini, "A quasi-Euclidean norm to speed up vector median filtering," IEEE Trans. Image Process., vol.9, no.10, pp.1704–1709, Oct. 2000.
[12] M. Barni, V. Cappellini, and A. Mecocci, "Fast vector median filter based on Euclidean norm approximation," IEEE Signal Process. Lett., vol.1, no.6, pp.92–94, June 1994.

[13] I.-C. Park and T.-H. Kim, "Multiplier-less and table-less linear approximation for square and square-root," Proc. IEEE Int'l Conf. Compuer Design, pp.373–383, Oct. 2009.
[14] B. Parhami, Computer arithmetic — Algorithms and hardware designs, Oxford University Press, 2000.
[15] M. Shammanna, S. Whitaker, and J. Canaris, "Cellular logic array for computation of squares," Proc. 3rd NASA Symp. VLSI Design pp.2.4.1–2.4.7, 1991.
[16] E.K.P. Chong and S.H. Zak, An introduction to optimization, 2nd ed., Wiley, 2001.
[17] Synopsys Co. Ltd., DesignWare IP, http://www.synopsys.com/IP, accessed June 6, 2010.
[18] Synopsys Co. Ltd., Design Compiler, http://www.synopsys.com/Tools/Implementation/RTLSynthesis, accessed June 6, 2010.
[19] ARM Ltd., ARM DSP instruction set extensions, http://www.arm.com/products/CPUs/cpu-arch-DSP.html, accessed April 1, 2010.
[20] IBM Co. Ltd., Power ISATM Version 2.05, http://www.power.org/resources/reading/PowerISA_V2.05.pdf, accessed April 1, 2010.

In-Cheol Park received the B.S. degree in electronic engineering from Seoul National University, Seoul, Korea, in 1986, the M.S. and Ph.D. degrees in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea, in 1988 and 1992, respecttively. Since June 1996, he has been an Assistant Professor and is now a Professor in the School of Electrical Engineering and Computer Science at KAIST. Prior to joining KAIST, he was with IBM T.J. Watson Research Center, Yorktown, NY, from May 1995 to May 1996, where he researched on high-speed circuit design. His current research interest includes computer-aided design (CAD) algorithms for high-level synthesis and VLSI architectures for general-purpose microprocessors. He received the Best Paper award at ICCD in 1999, and the best design award at ASP-DAC in 1997.

Tae-Hwan Kim received the B.S. degree in electrical engineering from Yonsei University, Seoul, Korea, in 2005, the M.S. degree in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea, in 2007. Currently, he is working toward the Ph.D. degree in the Department of Electrical Engineering and Computer Science at KAIST. His current research interest includes VLSI architectures for communication systems.