Multiplier-less and Table-less Linear Approximation for Square and Square-root

In-Cheol Park and Tae-Hwan Kim
Division of Electrical Engineering, Korea Advanced Institute of Science and Technology
icpark@ee.kaist.ac.kr , thkim@eeinfo.kaist.ac.kr

***Abstract*—Square and square-root 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 and square-root 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.**

I. 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 the square-related functions based on linear interpolation. 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 rest of the paper is organized as follows. In Section II, we briefly describe the previous works proposed to approximate square and square-root functions. In Section III, we propose the multiplier-less, table-less linear approximations for square function. Section IV explains how the proposed method can be applied to square-root function. In Section V, we evaluate the performance of the proposed approximations compared to those of the previous works, and then discuss the advantages of the proposed method. Concluding remarks are made in Section VI.

II. 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 products can be reduced by half [13]. This property has been actively utilized to approximate square functions in the estimation hardware such as Viterbi decoders. In [14], 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 $\{x_{N-1}, (N-1)\text{b}0\}$ and $x_{N-2:0}$ as in [14], 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 ($0 \le j \le i$), $\{x, y\}$ is the concatenation of x and y, and Nb0 is the N-bit binary representation of 0. Then x^2 can be expressed as $\{x_{N-1}, (N-1)\text{b}0\}^2 + 2\cdot\{x_{N-1},(N-1)\text{b}0\}\cdot x_{N-2:0} + (x_{N-2:0})^2$. By removing non-dominant terms, we can approximate x^2 as $\{x_{N-1},$

(N-1)b0}·({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 [13]. 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 [15]. 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-∞. However, a number of complicated operations such as division and multiplication are used in those approximations [11] [12].

III. PROPOSED APPROXIMATION FOR SQUARE FUNCTION

In this section, we propose a new approximation for the square function. The proposed method is to apply piecewise linear approximations. As shown in Fig. 1, the piecewise linear approximation partitions the input range 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. This linear interpolation is widely used to approximate complicated functions that should be computed with a large number of operations, and in general it requires tables to store the *initial offset* and the *slope* for each segment and a multiplier to do the *slope* multiplication. As the square operation is exactly computed with one multiplication, the piecewise linear interpolation seems to be inappropriate for the approximation of square-related functions. If we can remove the table and the multiplication, however, the linear interpolation can be a good candidate for efficient implementation.

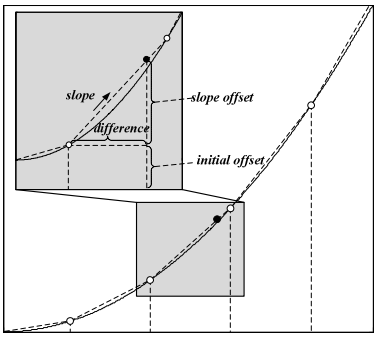

Fig. 1. General piecewise linear approximation.

The fundamental concepts behind the proposed method to approximate the square is as follows. First, if the input is a power of 2, say 2^k, where k is an integer not less than 0, the square is 2^{2k}. In other words, we can generate the square of 2^k by shifting the input left by k bits. If the input range is segmented to have boundaries at 2^k, we can eliminate the offset table. 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. The proposed approximation is developed under the above constraints to achieve a table-less and multiplier-less linear interpolation.

A. Proposed Linear Interpolation for Square Function

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

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

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

$$\text{slope of } \alpha = (2^{2i+1}-2^{2i})/(2^i+2^{i-1}-2^i)=2^{i+1}, \quad (2)$$

and

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

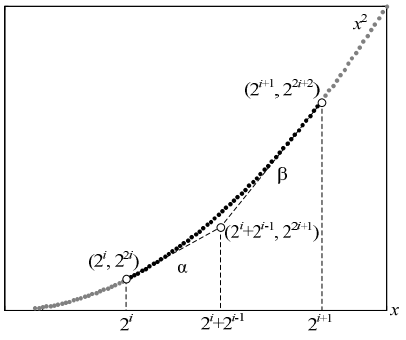

Fig. 2. Proposed piecewise linear approximation of x^2 for $2^i \le x < 2^{i+1}$.

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:

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

Alternatively, this can be expressed as

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

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:

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

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:

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

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

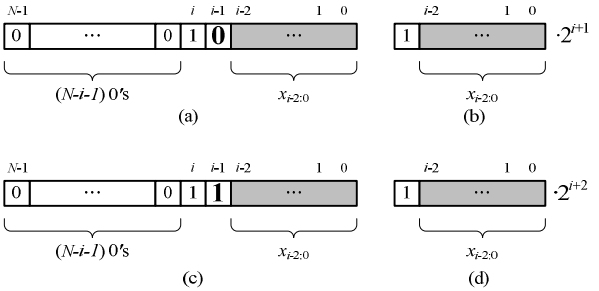

Fig. 3. Proposed method to calculate the approximate square. (a) $2^i \le x < 2^i+2^{i-1}$, (b) approximate square of (a), (c) $2^i+2^{i-1} \le x < 2^{i+1}$, and (d) approximate square of (c).

B. 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,

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

$RE(f_1(x))$ is maximized when x is equal to 2^i+2^{i-1}, which is calculated as

$$\begin{aligned} \mathrm{MAX}(RE(f_1(x))) &= \frac{\left| f_1(2^i+2^{i-1}) - (2^i+2^{i-1})^2 \right|}{(2^i+2^{i-1})^2} \\ &= \frac{\left| 2^{2i+1} - (2^i+2^{i-1})^2 \right|}{(2^i+2^{i-1})^2} = \frac{1}{9} \approx 0.1111 \end{aligned} \quad (9)$$

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

$$\mathrm{AVG}(RE(f_1(x))) = \left(\sum_{x=1}^{2^N-1} RE(f_1(x)) \right) / \left(2^N - 1 \right). \quad (10)$$

The average relative errors in (10) 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 constancy of the relative error 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, $f_2(x) = f_1(x)+(f_1(x)\cdot 2^{-4})$. Fig. 4 illustrates the relative errors resulting from $f_2(x)$ and $f_1(x)$ for the i-th segment, where we can see that

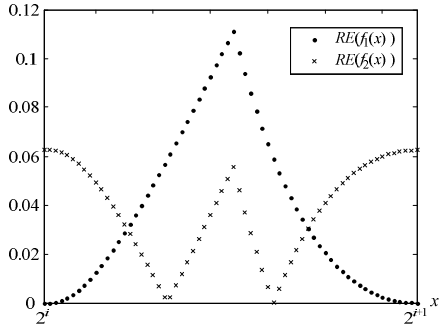

Fig. 4. Relative error of the proposed approximate square functions for $2^i \le x < 2^{i+1}$.

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

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.

IV. PROPOSED APPROXIMATION FOR SQUARE-ROOT FUNCTION

In this section, we propose a new approximation for the square-root function, and analyze its error performance. The proposed method is also based on piecewise linear approximations, and does not need any multiplications and tables.

A. 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. 5. At the middle point in this segment, 2^{2i+1}, its square-root can be approximated as

$$2^{(2i+1)/2} = \sqrt{2}\cdot 2^i \approx (1+1/2)\cdot 2^i = 2^i + 2^{i-1}. \quad (11)$$

By taking (11), the curve of $x^{1/2}$ in the i-th segment can be linearly approximated with two lines, γ and δ, as depicted in Fig. 5. Let $f_3(x)$ be the proposed linear interpolation of the square-root of x. Then, $f_3(x)$ can expressed with a number of line equations as follows:

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

If x is less than 2^{2i+1}, x-2^{2i} in (12) can be replaced with $x_{2i\text{-}1:0}$, and if x is less than 2^{2i+2}, x-2^{2i+1} in (12) can be replaced with $x_{2i:0}$. Fig. 6 (a) and Fig. 6 (c) correspond to these cases, respectively. In (12), 2^{2i+1}+$x_{2i\text{-}1:0}$ and 2^{2i+2}+2^{2i+1}+$x_{2i:0}$ can be achieved by doing simple concatenations, {2b10, $x_{2i\text{-}1:0}$} 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 $x_{2i+1:2i}$, $f_3(x)$ can be expressed as follows,

$$f_3(x) = \begin{cases} \{2b10, x_{2i-1:0}\}\cdot 2^{-i-1} & \text{for } x_{2i+1} = 0 \\ \{2b11, x_{2i:0}\}\cdot 2^{-i-2} & \text{for } x_{2i+1} = 1 \end{cases}. \quad (13)$$

The proposed square-root approximations corresponding to Fig. 6 (a) and Fig. 6 (c) are shown in Fig. 6 (b) and Fig. 6 (d), respectively.

B. Error Analysis and Error Compensation of the Approximate Square-root

The relative error of $f_3(x)$ is defined below:

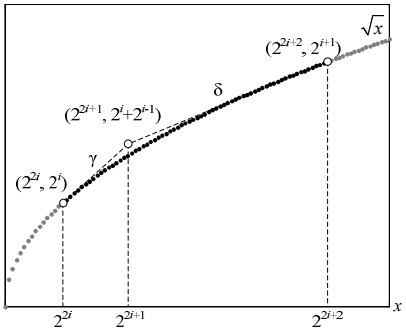

Fig. 5. Proposed piecewise linear approximation of $x^{1/2}$ for $2^{2i} \le x < 2^{2i+2}$.

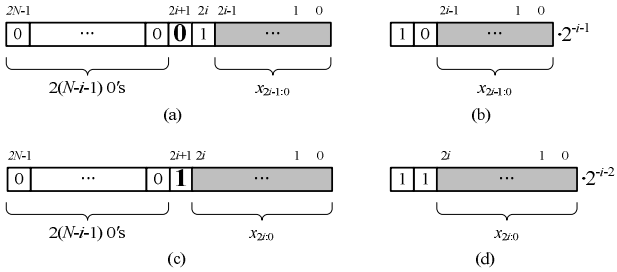

Fig. 6. Proposed method to calculate the approximate square-root. (a) $2^{2i} \le x < 2^{2i+1}$, (b) approximate square-root of (a), (c) $2^{2i+1} \le x < 2^{2i+2}$, and (d) approximate square root of (c).

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

Bit-width of x	MAX($RE(f_2(x))$)	AVG($RE(f_2(x))$)	MAX($RE(f_2(x))$)/ MAX($RE(f_1(x))$)	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 square-root

Bit-width of x	MAX($RE(f_3(x))$)	AVG($RE(f_3(x))$)
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

$$RE(f_3(x)) = \left| f_3(x) - \sqrt{x} \right| / \sqrt{x} \,. \qquad (14)$$

The relative error is maximized when x is 2^{2i+1}, which is calculated as

$$\mathrm{MAX}(RE(f_3(x))) = \frac{3/2 - \sqrt{2}}{\sqrt{2}} \approx 0.0607 \,. \qquad (15)$$

Similar to (10), the average relative error of $f_3(x)$ is defined as

$$\mathrm{AVG}(RE(f_3(x))) = \left(\sum_{x=1}^{2^{2N}-1} RE(f_3(x)) \right) / \left(2^{2N} - 1 \right) . \qquad (16)$$

The average relative errors evaluated for various bit-widths are listed in Table 3 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.

$f_3(x)$ 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 3. To reduce the maximum error, we can apply a compensation technique similar to that used in the previous section. As $f_3(x)$ is always larger than $x^{1/2}$, we can scale down $f_3(x)$ by a factor of 31/32 to achieve the smaller maximum relative error. Let $f_4(x)$ be the compensated one; that is, $f_4(x)=f_3(x)-(f_3(x)\cdot 2^{-5})$. Fig. 7 compares the error performances of $f_3(x)$ and $f_4(x)$. Applying the proposed compensation reduces the maximum relative error to almost a half, while maintaining the average relative error performance. In Table 4, the maximum and average relative errors of $f_4(x)$ are listed for various bit-widths.

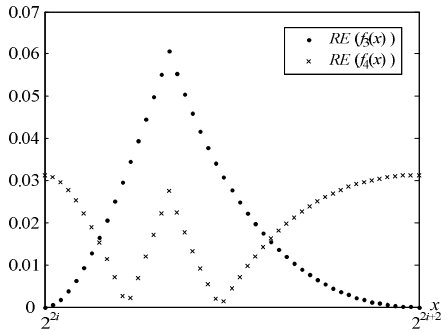

Fig. 7. Relative error of the proposed approximate square-root with the error compensation for $2^{2i} \le x < 2^{2i+2}$.

V. COMPARISON AND DISCUSSION

In Fig. 8 and Fig. 9, 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, $f_1(a) \ge f_1(b)$, $f_2(a) \ge f_2(b)$, $f_3(a) \ge f_3(b)$, and $f_4(a) \ge f_4(b)$ if $a \ge b$. These monotonic properties are significant in comparing two approximated values.

The proposed approximations for square and square-root functions can be understood by employing 1st-order Taylor series expansions. For example, the proposed approximate square in (4) is the same as the composition of the 1st-order Taylor expansions for $2^i \le x < 2^i + 2^{i-1}$ and $2^i + 2^{i-1} \le x < 2^{i+1}$. However, note that the proposed methods have a contribution in selecting the specific evaluation points of the Taylor expansions that result in multiplier-less and table-less implementations. Moreover, simple calculation methods are derived as expressed in (7) and (13).

The proposed multiplier-less, table-less linear interpolation applied to approximate the square-related functions 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

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

Bit-width of x	MAX($RE(f_4(x))$)	AVG($RE(f_4(x))$)	MAX($RE(f_4(x))$)/ MAX($RE(f_3(x))$)	AVG($RE(f_4(x))$)/ AVG($RE(f_3(x))$)
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

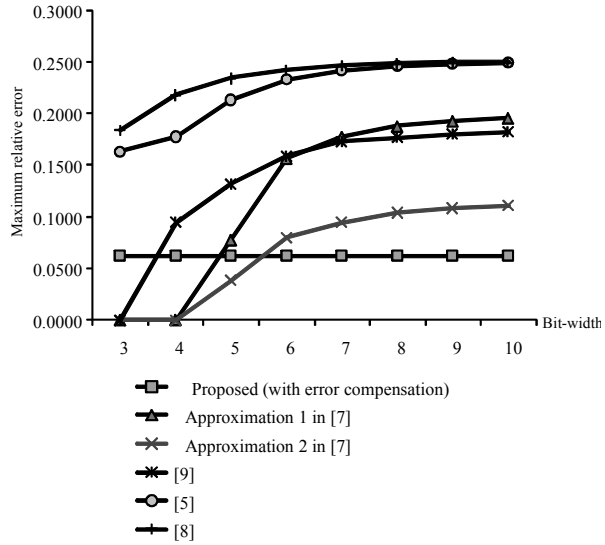

Fig. 8. Comparison of the maximum relative errors of square approximations.

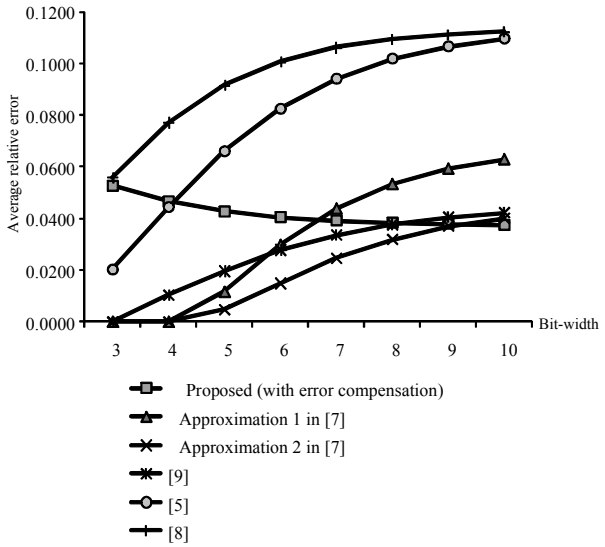

Fig. 9. Comparison of the average relative errors of square approximations.

supported in most of contemporary processor architectures, as specified in [16] and [17]. Hence, the proposed approximations can be efficiently implemented with the assistance of those pre-defined instructions.

VI. CONCLUSION

In this paper, we have proposed new methods to approximate the square-related functions. 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 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.

ACKNOWLEDGMENT

This work was supported by Korean Intellectual Property Office (KIPO) and by IC Design Education Center (IDEC).

REFERENCES

[1] M. R. Soleymani, S.D. Morgera, "A Fast MMSE Encoding Technique for Vector Quantization," *IEEE Trans. on Communications*, 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. on Information Theory*, vol. 13, no. 4, pp. 260-269, April 1967.

[3] Hochwald. B. M., Ten Brink. S., "Achieving Near-capacity on a Multiple-antenna Channel," *IEEE Trans. on Communications*, vol. 51, no. 3, pp. 389-399, March 2003

[4] J. Astola, P. Haavisto, Y. Neuvo, "Vector Median Filter," *in Proc. of IEEE*, vol. 78, no. 4, pp. 690-710, April 1990.

[5] Hiasat, A.A. and Abdel-Aty-Zohdy, H.S., "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] Jae-Tack Yoo, Kent F. Smith, Ganesh 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. Pierre Langlois, Dhamin Al-Khalili, "Carry-Free Approximate Squaring Functions with $O(n)$ complexity and $O(1)$ Delay," *IEEE Trans. on Circuits and Systems-II: Express Briefs*, vol. 53, no. 5, pp. 374-378, May 2006.

[8] Aria Eshraghi, Terri S. Fiez, Kel D. Winters, Thomas R. Fischer, "Design of a New Squaring Function for the Viterbi Algorithm," *IEEE J. Solid-State Circuits*, vol. 29, no. 9, pp. 1102-1107, Sep. 1994.

[9] Ming-Hwa Sheu, Su-Hon Lin, "Fast Compensative Design Approach for the Approximate Squaring Function," *IEEE J. Solid-State Circuits*, vol. 37, no. 1, Jan. 2002.

[10] Changkyu Seol, Kyungwhoon Cheun, "A Low Complexity Euclidean Norm Approximation," *IEEE Trans. on Signal Processing*, vol. 56, no. 4, pp. 1721-1726, April 2008.

[11] Mauro Barni, Fabio Buti, Franco Bartolini, Vito Cappellini, "A Quasi-Euclidean Norm to Speed Up Vector Median Filtering," *IEEE Trans. on Image Processing*, vol. 9, no. 10, pp. 1704-1709, Oct. 2000.

[12] Mauro Barni, Vito Cappellini, A. Mecocci, "Fast Vector Median Filter Based on Euclidean Norm Approximation," *IEEE Signal Processing Letter*, vol. 1, no. 6, pp. 92-94, June 1994.

[13] Behrooz Parhami, "Computer Arithmetic - Algorithms and Hardware Designs," Oxford University Press 2000.

[14] M. Shammanna, S. Whitaker, J. Canaris, "Cellular Logic Array for Computation of Squares," *3rd NASA Symposium on VLSI Design 1991*, pp.2.4.1-2.4.7.

[15] Edwin K.P. Chong, Stani H. Zak, "An Introduction to Optimization - 2nd Edition," WILEY 2001.

[16] ARM Ltd., ARM DSP Instruction Set Extensions, http://www.arm.com/products/CPUs/cpu-arch-DSP.html

[17] IBM Co. Ltd., Power ISATM Version 2.05, www.power.org/resources/reading/PowerISA_V2.05.pdf