electronics

MDPI

Article

Piecewise Parabolic Approximate Computation Based on an Error-Flattened Segmenter and a Novel Quantizer

Mengyu An [1], Yuanyong Luo [2,*], Muhan Zheng [1], Yuxuan Wang [1], Hongxi Dong [1], Zhongfeng Wang [1], Chenglei Peng [1,*] and Hongbing Pan [1]

[1] School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China; anmengyu@smail.nju.edu.cn (M.A.); mf1923101@smail.nju.edu.cn (M.Z.); wangyuxuan@nju.edu.cn (Y.W.); mf1923117@smail.nju.edu.cn (H.D.); zfwang@nju.edu.cn (Z.W.); phb@nju.edu.cn (H.P.)
[2] Huawei Corporation, Bantian, Longgang District, Shenzhen 518116, China
* Correspondence: luoyuanyong@yeah.net (Y.L.); pcl@nju.edu.cn (C.P.)

Abstract: This paper proposes a novel Piecewise Parabolic Approximate Computation method for hardware function evaluation, which mainly incorporates an error-flattened segmenter and an implementation quantizer. Under a required software maximum absolute error (MAE), the segmenter adaptively selects a minimum number of parabolas to approximate the objective function. By completely imitating the circuit's behavior before actual implementation, the quantizer calculates the minimum quantization bit width to ensure a non-redundant fixed-point hardware architecture with an MAE of 1 unit of least precision (ulp), eliminating the iterative design time for the circuits. The method causes the number of segments to reach the theoretical limit, and has great advantages in the number of segments and the size of the look-up table (LUT). To prove the superiority of the proposed method, six common functions were implemented by the proposed method under TSMC-90 nm technology. Compared to the state-of-the-art piecewise quadratic approximation methods, the proposed method has advantages in the area with roughly the same delay. Furthermore, a unified function-evaluation unit was also implemented under TSMC-90 nm technology.

Keywords: error-flattened; segmenter; quantizer; multifunctional unit

Citation: An, M.; Luo, Y.; Zheng, M.; Wang, Y.; Dong, H.; Wang, Z.; Peng, C.; Pan, H. Piecewise Parabolic Approximate Computation Based on an Error-Flattened Segmenter and a Novel Quantizer. *Electronics* **2021**, *10*, 2704. https://doi.org/10.3390/electronics10212704

Academic Editor: Maciej Ławryńczuk

Received: 26 September 2021
Accepted: 3 November 2021
Published: 5 November 2021

1. Introduction

High speed hardware function evaluation has been widely deployed in computer graphics. Graphical processing units (GPU) use special function units (SFU) to compute elementary functions, which are essential operations in graphical processing [1]. As the constraints on performance become much more stringent, more and more hardware-oriented methods for high-speed elementary functions have been proposed. Generally, implementation methods of unary functions can be separated into two categories: iterative methods and non-iterative methods. From the perspective of VLSI implementation, iterative methods need to repeat the same or similar calculations multiple times on the time axis, while non-iterative methods use the calculation resource only once.

The mainstream iterative methods of complex computing units are the Newton–Raphson (NR) method [2] and Goldschmidt algorithms [3], which are multiplicative-based. Although these methods converge quickly, each iteration requires several multipliers, which lead to a long execution time and large area. Coordinate Rotation Digital Computer (CORDIC) is also popular [4,5], but its latency is too high due to its multicycle execution delays.

Non-iterative methods are usually based on look-up tables (LUTs). According to the size of LUTs and computation complexity, table-based methods are further divided into three categories: computer-bound methods, table-bound methods, and in-between methods. Computer-bound methods use small-sized LUTs to store significant parameters

which are used in cubic or higher-degree polynomial calculation. Therefore, a lot of multiplications and additions are necessary, which leads to a high hardware overhead. The polynomial is usually implemented with fused multiply-add units using Horner's rule. Table-bound methods use large LUTs and simple calculation units, usually a few additions. Bipartite table methods (BTM) [6] and multipartite table methods [7] are typical examples of table-bound methods, which use a few tables and some additions. In general, table-bound methods are suitable for low-precision applications, because as the accuracy increases, the size of the LUTs increases exponentially. In high-precision applications, the hardware overhead of large LUTs is unaffordable.

In middle-to-high-precision applications, in-between methods are more popular, because they obtain a better trade-off between the size of LUTs and arithmetic complexity. This method can be further subdivided into linear approximation and quadratic approximation depending on the polynomial approximation degree. Linear approximation has advantages when the accuracy is low, because of its high speed and low computation complexity [8,9]. It is usually used in low-precision applications or to provide an initial value for other methods. Ref. [10] proposed two levels of approximation for medium-to-high-precision applications. In the first level of initial approximation, piecewise degree-one polynomials are used to approximate the target function, and approximations to the corresponding difference functions are generated. Afterward, in the second level of refined approximation, the shared normalized difference function is computed using either direct LUTs or another piecewise interpolation. This method reduces the total area at the cost of a long delay. In general, quadratic approximation offers a better compromise between LUT size and computation complexity in medium-to-high-precision applications [11,12].

The uniform segmentation method is a common method in quadratic approximation. The input X is split into two parts, the most significant part X_1 and the least significant part X_2. The target function is evenly divided into 2^{X_1} subintervals and approximated by a quadratic polynomial $aX_2^2 + bX_2 + c$ for each subinterval. The optimal quadratic polynomial for a special subinterval can be obtained by interpolation at Chebyshev points [13] or minimax approximation [14], which needs Maple toolbox. In hardware implementation, X_1 can be used to index the coefficients a, b, c directly. Many works have adopted this method and focus on reducing the coefficients bit width and optimizing the multiplier. Ref. [15] used an enhanced minimax approximation to reduce the LUT size, which takes into account the effect of rounding the coefficients to a finite size through an iterative process. Ref. [16] proposed a novel technique to minimize polynomial coefficients bit width, which uses integer linear programming (ILP) to optimize the polynomial coefficients, considering all error components simultaneously. The multiplier in [17] is optimized in detail based on [16]. Uniform segmentation method has two obvious advantages. Firstly, the index logic is simple because X_1 contains the information of each subinterval. Secondly, the multiplier bit width is small. On the one hand, X_2 has less bit width than X. On the other, the coefficients can be reduced by novel methods. However, the accuracy of such methods is limited by the largest error among all segments. In high-precision applications, the number of segments is too large, leading to an LUT size that is too large and just unnecessary. When evaluating functions with high nonlinearity, uniform segmentation methods are not efficient. As for SFU in GPU, multiple functions need to be implemented, the shortcomings caused by large LUTs are much more apparent.

Therefore, several non-uniform segmentation methods have been proposed [18–23]. Refs. [18,19] proposed multilevel hierarchical segmentation methods, in which the segment size at each level can be the same or different by increasing or decreasing the order of power-of-two. Ref. [20] presented another similar segmentation method targeting for floating-point arithmetic which places the boundary of segments so that every segment contains consecutive numbers, and thus the segment index encoder can be realized by simple combinational logic. These methods are not flexible enough to dynamically fit target functions with high nonlinearity, although the hardware design of the segment index

encoder is simple. In [21], the segmentation boundary could be at any position of the interval depending on the adopted segmentation schemes. However, the design of the segment index encoder is complicated. Refs. [22,23] presented new non-uniform segmentation approaches that merge a fixed number of uniform base segments into a larger non-uniform segment and design an innovative method to remap addresses. However, this method does not cause the number of segments to reach the theoretically minimum value, and there is still room for improvement.

The SFU of GPU usually needs the implementation of multiple functions, so the size of LUTs is significant to the whole area. This paper proposes a new method, which cuts down the LUTs' size by reducing the number of segments to the theoretical minimum value for any given precision.

An error-flattened segmenter and a novel quantizer are components of the piecewise parabolic approximate computation method. The segmenter uses piecewise parabolas to approximate the objective function under the required software maximum absolute error (MAE), and ensures that the required accuracy is achieved with the minimum number of segments. In detail, we discretize the data and set a target value as the unified software MAE for each segment. In each segment, we guarantee the real MAE to be at the minimum. Then, the segmenter will select the maximum range for each segment and derive the corresponding coefficients until the whole input range has been covered. Because the range of each region has been maximized, the number of segments is minimized. So far, we have used piecewise parabolas with the lowest number of segments to approximate the target function in the input region. Based on the segmenter, we design a general hardware architecture. The quantizer is designed for this hardware architecture. It can completely imitate the input and output behavior of the circuits and ensure that the hardware overhead is minimized with an MAE of 1 unit of least precision (ulp). Therefore, the designer does not have to adjust the hardware bit width iteratively.

The main contributions of this paper are summarized as follows:

- The proposed method reduces the number of segments to the theoretical minimum value in any given precision to reduce the size of the LUTs, which has obvious advantages in reducing the area of applications that implement multiple functions simultaneously, such as SFU.
- The proposed arithmetic circuits can be implemented without a waste of hardware resources. With the help of an innovative quantizer, we quantize the coefficients to the minimum bit width, which can guarantee the precision up to 1 ulp.
- It is a calculation method with an absolutely controllable error. The exact bits of the result in hardware can be set in advance.
- The proposed method is applicable to all unary functions with finite codomain.

The remainder of this paper is organized as follows. Section 2 depicts the basic principles of the proposed method. The error-flattened segmenter is introduced and verified in Section 3. In Section 4, the hardware architecture and the quantizer are described. The details of hardware implementation and comparison with the existing methods are demonstrated in Section 5. Finally, Section 6 summarizes our work.

2. Basic Principles

2.1. Basis of Piecewise Parabola Approximation

Let the objective function be expressed as $\mathrm{f(x)}, \mathrm{x} \in [\mathrm{M}, \mathrm{N}]$. Piecewise parabola approximation uses a special parabola segmenter to split the input range $\mathrm{x} \in [\mathrm{M}, \mathrm{N}]$ into several segments. Each segment is approximated by a parabola $h_i(x) = a_i x^2 + b_i x + c_i$. The coefficients a_i, b_i, c_i are stored in the LUT in advance. Over the entire input range, the function can be approximated as

$$\mathrm{f(x)} \approx \mathrm{h(x)} = \{h_i(x)\}_{i=1,2,3,\ldots} = \{a_i x^2 + b_i x + c_i\}_{i=1,2,3,\ldots}. \quad (1)$$

Four sets of data need to be determined when using the proposed method. The first is the segment start point value in the input range $\{x_i\}_{i=1,2,3,\ldots}$. The others are the quadratic term coefficients $\{a_i\}_{i=1,2,3,\ldots}$, linear term coefficients $\{b_i\}_{i=1,2,3,\ldots}$, and constant term coefficients $\{c_i\}_{i=1,2,3,\ldots}$. Supposing the objective function has n segments in the input range, n − 1 start point values, n quadratic term coefficients, n linear term coefficients, and n constant term coefficients need to be stored in the LUT.

2.2. Discretization

The data are discrete in fixed-point calculations in VLSI. Let the objective function be $f(x), x \in [M, N]$. Assume that the number of fractional bits of x in hardware is Q. We sample the input range [M,N] with an equal interval of 2^{-Q}. According to Equation (2), the input becomes a one-dimensional vector with the size of NUM:

$$xq = xq(1{:}\,NUM) = \left\{M, M + \frac{1}{2^Q}, M + \frac{2}{2^Q}, \ldots, N - \frac{1}{2^Q}, N\right\}. \quad (2)$$

At this time, the output fq(x) also becomes a one-dimensional vector with the size of NUM:

$$fq = fq(1{:}\,NUM) = \{f(xq(1)), f(xq(2)), \ldots, f(xq((NUM))\}. \quad (3)$$

The design of parabola segmenter is based on the discrete point set {xq(i),fq(i)}.

2.3. Metric for Error Evaluation

The maximum absolute error (MAE) is a common metric to evaluate the approximate error. Considering the fixed-point implementation with Q fractional bits in hardware, the continuous input range [M,N] is sampled into NUM points:

$$NUM = (N - M) \times 2^Q + 1. \quad (4)$$

In hardware implementation, xq, fq(x), and h(x) are one-dimensional vectors with the size of NUM. MAE is the absolute value of the worst-case approximate error. It is defined as

$$MAE = \max |fq(xq) - h(xq)|. \quad (5)$$

3. Error-Flattened Segmenter

3.1. Minimization of MAE

In this subsection, we explore ways to minimize MAE for a given segment. Suppose the input of the segment is $xq(j{:}\,k), 1 \le j < k \le NUM$. Ref. [14] pointed out that interpolation at cleverly chosen points could be a sensible solution: interpolating a function at Chebyshev points gives a polynomial almost as good as the minimax polynomial. The parabola generated by Chebyshev interpolation can be directly calculated by MATLAB, while the minimax method needs a Maple toolbox. Thus, with the aim of minimizing MAE, we choose three Chebyshev points of the target function and fine-tune it to generate the optimal parabola [24]. As for n-order interpolation, the Chebyshev nodes on [−1, 1] are:

$$u_i = \cos\left(\frac{(2i + 1)\pi}{2N}\right), i = 0,1,2, \ldots, N - 1. \quad (6)$$

Therefore, the third-order Chebyshev nodes on [−1, 1] are:

$$\begin{cases} u_0 = \cos\left(\frac{\pi}{6}\right) = \frac{\sqrt{3}}{2} \\ u_1 = \cos\left(\frac{3\pi}{6}\right) = 0 \\ u_2 = \cos\left(\frac{5\pi}{6}\right) = -\frac{\sqrt{3}}{2}. \end{cases} \quad (7)$$

The Chebyshev nodes are then transformed from [−1,1] to [a, b] by the following formula:

$$v_i = u_i \cdot \frac{b-a}{2} + \frac{a+b}{2}. \quad (8)$$

The three Chebyshev nodes in the i-th subinterval $xq(j:k)$ become:

$$\begin{cases} x_{is} = -\frac{\sqrt{3}}{2} \cdot \frac{xq(k) - xq(j)}{2} + \frac{xq(k) + xq(j)}{2} \\ x_{im} = \frac{xq(k) + xq(j)}{2} \\ x_{ie} = \frac{\sqrt{3}}{2} \cdot \frac{xq(k) - xq(j)}{2} + \frac{xq(k) + xq(j)}{2}. \end{cases} \quad (9)$$

In order to show more clearly the relationship between the Chebyshev nodes and xq(j), xq(k), Equation (9) can be rewritten as

$$\begin{cases} x_{is} = c1 \cdot \big(xq(k) - xq(j)\big) + xq(j) \\ x_{im} = c1 \cdot \big(xq(k) - xq(j)\big) + xq(j) \\ x_{ie} = c3 \cdot \big(xq(k) - xq(j)\big) + xq(j). \end{cases} \quad (10)$$

where c1 is $(-\sqrt{3}/2 + 1/2)$, c2 is $1/2$, c3 is $(\sqrt{3}/2 + 1/2)$.

The target function values of x_{is} x_{im} and x_{ie} are expressed as y_{is}, y_{im}, and y_{ie}. The coefficients of the parabola in the i-th subinterval are determined by these three points:

$$\begin{cases} b_i = \frac{(x_{im}^2 - x_{ie}^2)(y_{is} - y_{im}) - (x_{is}^2 - x_{im}^2)(y_{im} - y_{ie})}{(x_{im}^2 - x_{ie}^2)(x_{is} - x_{im}) - (x_{is}^2 - x_{im}^2)(x_{im} - x_{ie})} \\ a_i = \frac{\big(y_{is} - y_{im} - b_i(x_{is} - x_{im})\big)}{(x_{is}^2 - x_{im}^2)} \\ c_i = y_{is} - a_i . x_{is}^2 - b_i x_{is.} \end{cases} \quad (11)$$

We plot the error distribution diagram of the first segment for function $\ln(1 + x)$ in Figure 1. As can be seen from Figure 1a, there are four error peaks, and the absolute values of four peaks are almost equal. However, there is still a small gap between the four peaks. Ref. [14] pointed out that p^* is the minimax degree-2 approximation to function f on [a,b] if and only if there exist at least four values $x_0, x_1, x_2, x_3 \in [a, b]$, such that $|p^*(x_i) - f(x_i)| = \|f - p^*\|_\infty$. Therefore, we need to make the four peaks equal. The first error peak and the last error peak are taken as the benchmark. Through the error distribution diagram, we can see that the second peak is slightly larger and the third peak is slightly smaller. Therefore, we only need to move x_{im} to the right—that is, to increase c2. As shown in Figure 1b, when c2 becomes 0.501, the four peaks are equal. For other functions, we can perform the same operation to make the four error peaks in the subinterval equal, as shown in Figure 2. The fine-tuned c2 can be expressed as $c2' = c2 + \delta$. The δ for a few common functions are shown in Table 1.

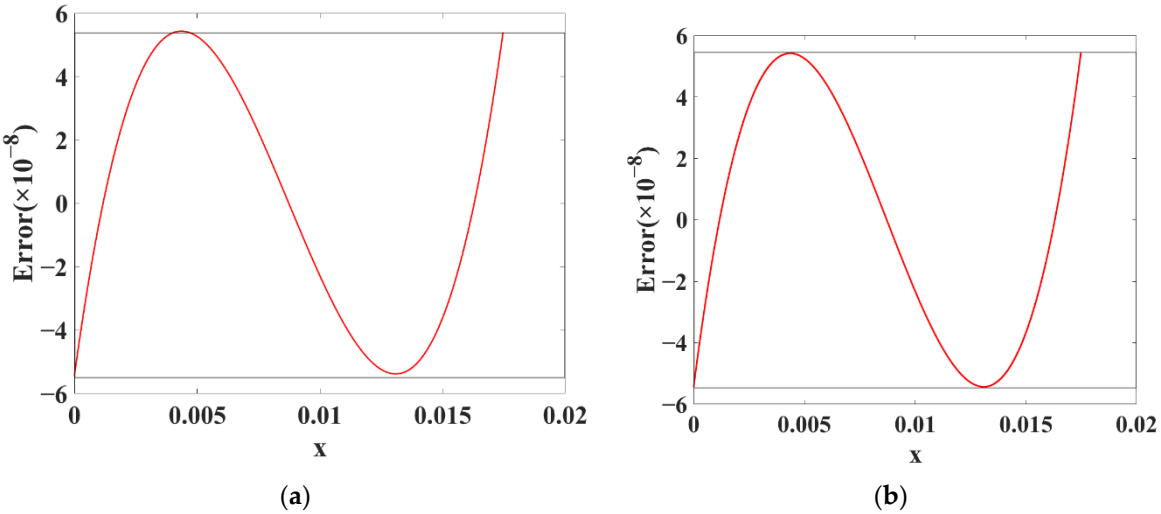

Figure 1. Error distribution of the first segment for function ln(1 + x). (**a**) ln(1 + x), c2 = 0.500; (**b**) ln(1 + x), c2 = 0.498.

Table 1. The δ for common functions.

Function	δ	Function	δ
2^x	0.01	sin(1+x)	0
$\sqrt{1+x}$	−0.02	cos(1+x)	0
$\frac{1}{1+x}$	−0.02	$log_2(1+x)$	−0.01
$\frac{1}{\sqrt{1+x}}$	−0.02	e^{1+x}	0
ln(1+x)	−0.02	2^{-x}	−0.01

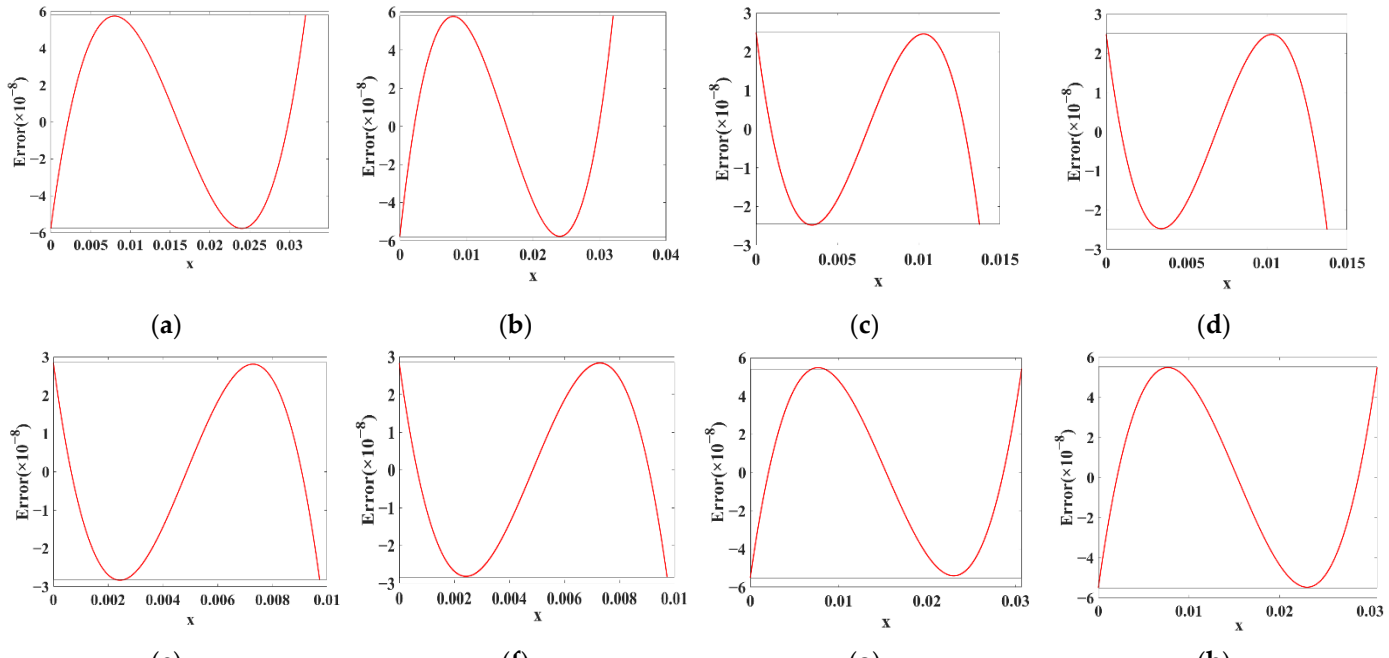

Figure 2. Error distribution of the first segment. (**a**) e^{1+x}, c2 = 0.500; (**b**) e^{1+x}, c2 = 0.501; (**c**) $1/\sqrt{1+x}$, c2 = 0.500; (**d**) $1/\sqrt{1+x}$, c2 = 0.498; (**e**) $1/(1+x)$, c2 = 0.500; (**f**) $1/(1+x)$, c2 = 0.498; (**g**) $\sqrt{1+x}$, c2 = 0.500; (**h**) $\sqrt{1+x}$, c2 = 0.498.

3.2. Minimization of the Segments Number with a Given Precision

Here, we propose an innovative segmentation method, which can adaptively generate the minimum number of segments under a given precision and a certain input range.

First of all, the input range of a segment needs to be maximized with a given MAE. The process is easy to understand. From the previous section, we can calculate the minimum MAE of a certain segment. If the MAE is greater than the predefined error, it means that the segment width is too large and should be reduced; if the MAE is no greater than the predetermined error, it means that the segment width may be too small, and there may be room for enlargement.

Assume that the required MAE of the software segmenter is MAE_sw and the objective function is $\{xq(i), fq(i)\}_{i=1:NUM}$. Then, suppose that the function in $xq(1:j-1)$ has been successfully approximated and the function in $xq(j:NUM)$ remains to be addressed. Now, we need to determine the input range of the next segment in $xq(j:NUM)$. The start point is $xq(j)$ and the end point $xq(ep)$ needs to be found. In theory, there are (NUM-j) possibilities for the selection of the end point $xq(ep)$, and the exhaustion method can be used to select it. According to Equation (6), the MAE for every choice of $xq(ep)$ can be calculated and expressed as $MAE(j+1:NUM)$. Among those MAEs which are no greater than the MAE_sw, we can choose the largest corresponding segment. However, the calculation of the exhaustion method is time consuming, resulting in the segmenter running on the software for hours or even days. Therefore, we optimize the calculation process of the segmenter based on the method proposed in [9].

We design the calculation process of segment width maximization based on dichotomy, as depicted in Figure 3. At initialization time, we assign the start index number j of the unsegmented input $xq(j:NUM)$ to the segment start pointer sp and left pointer of the dichotomy window lp. Similarly, we assign the end index number NUM of the unsegmented input $xq(j:NUM)$ to the segment end pointer ep and right pointer of the dichotomy window rp.

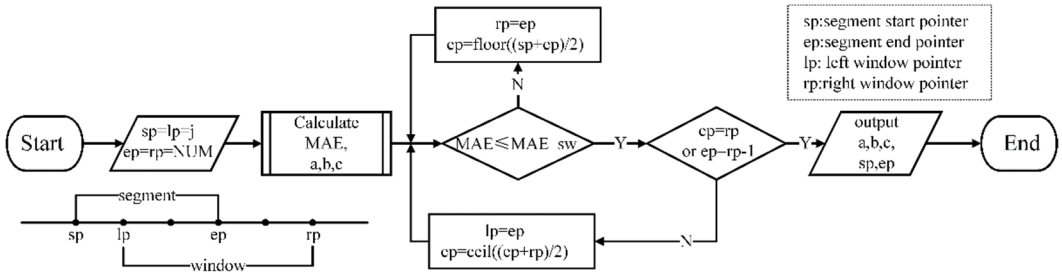

Figure 3. Calculation process of segment width maximization based on dichotomy.

Then, the minimum MAE as well as the coefficients of the approximate parabola in the input range $xq(sp:ep)$ are calculated by the method introduced previously. If the MAE satisfies the required precision MAE_sw and one of the maximum width judgment conditions $ep = rp$ is true, then the corresponding coefficients, segment start pointer sp, and segment end pointer ep at this time are outputs. If the first calculated MAE does not meet the accuracy requirements, then update the right window pointer rp to the current end pointer ep, and move the end pointer ep to the left by half of $xq\ (lp:ep)$ and recalculate MAE.

It can be seen that, after the right window is shifted to the left, the updated indexes from rp to NUM cannot be used as the end pointer ep. Then, the judgment condition of the maximum segmentation end point that satisfies the calculation precision requirements should be $ep = rp - 1$—that is, the previous point of rp cannot meet the requirements. Therefore, even if MAE meets the precision requirement, it still necessary to judge whether the segment width $xq(sp:ep)$ at this time is the largest according to whether ep = rp − 1 is true. If it does not satisfy the judgment, the left window lp needs to be updated

to the current end pointer ep, then the end point ep moves to the right by half of $xq(ep: rp)$ and MAE is recalculated. After repeated iterations of the above calculation process, the judgment condition $ep = rp - 1$ will finally hold and the corresponding coefficients, segment start pointer sp, and segment end pointer ep will be the outputs.

The segmentation in the whole input range can be carried out as in the flowchart in Figure 4. First of all, the calculation process of segment width maximization is adopted to select the first segment $xq(1: ep)$. The start pointer, the end pointer, and the corresponding coefficients are stored. Then, the segment start pointer sp is updated to ep+1, and the segment end pointer ep is updated to NUM. The calculation of the segment width maximization is continuously performed until the sp points to NUM+1. Now, segmentation over the whole input range has been completely discussed. Assuming the number of the segments is n, we put the start pointer sp of every segment together to get a vector of start pointers:

$$S = \{sp\} = \{S_1, S_2, S_3, \ldots, S_n\}. \tag{12}$$

Similarly, we can obtain the vectors of quadratic coefficients, linear coefficients, and constant coefficients:

$$\begin{cases} a = \{a_1, a_2, a_3, \ldots, a_n\} \\ b = \{b_1, b_2, b_3, \ldots, b_n\} \\ c = \{c_1, c_2, c_3, \ldots, c_n\}. \end{cases} \tag{13}$$

Up until now, the objective function $fq(1: NUM)$ can be approximated by piecewise parabolas:

$$fq \approx \begin{cases} a_1x_1^2 + b_1x_1 + c_1, x_1 \in \left[xq(1), xq(S_2)\right) \\ a_2x_2^2 + b_2x_2 + c_2, x_2 \in \left[xq(S_2), xq(S_3)\right) \\ a_3x_3^2 + b_3x_3 + c_3, x_3 \in \left[xq(S_3), xq(S_4)\right) \\ \ldots \\ a_nx_n^2 + b_nx_n + c_n, x_n \in [xq(S_n), xq(NUM)]. \end{cases} \tag{14}$$

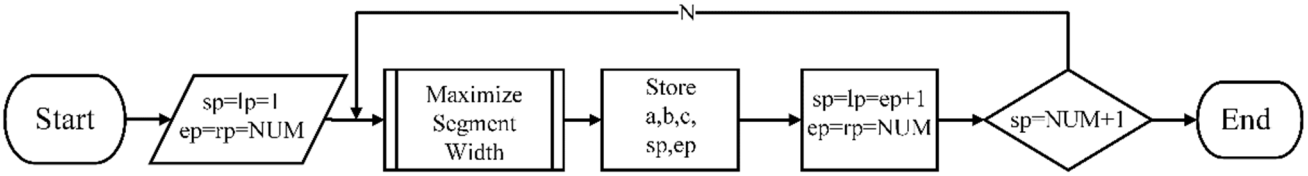

Figure 4. Segmentation process of the whole input range.

3.3. Test of Segmenter Performance

We test whether the segmenter meets the preset performance in MATLAB. The discretization interval in Equation (2) is set as 2^{-20}. The error distribution curve of square root $\sqrt{x}$ with 12 segments is shown in Figure 5. The preset MAE of software segmenter MAE_sw is 4.502×10^{-7}. As can be seen, the maximum absolute error of each segment is equal. There are four peaks per segment in the error distribution diagram, whose absolute values are all equal to MAE_sw. Thus, the MAE in each segment is minimized and error flattening in the whole input range is realized. In conclusion, the number of segments reaches the theoretical minimum. In addition, the MAE_sw is set in advance; this reflects that the error is absolutely controllable. Experiments show that the designed segmenter achieves basic functionality.

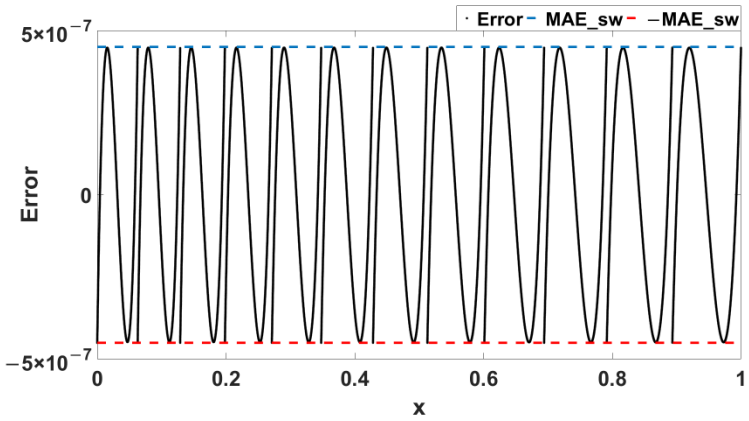

Figure 5. Error distribution of square root $\sqrt{x}$ with 12 segments in [0, 1).

Furtherly, we set the discretization interval in Equation (2) as 2^{-23}. For different target functions, such as 2^x, $\log_2^{1+x}$, $\sin(x)$, $1/(1+x)$, we set different MAE_sw and plot the approximate function and error distribution in Figure 6. As is shown in Figure 6, for each segment of each function, there are four peaks whose absolute values equal MAE_sw. According to [14], the MAE in each segment is minimized and error flattening in the whole input range is realized.

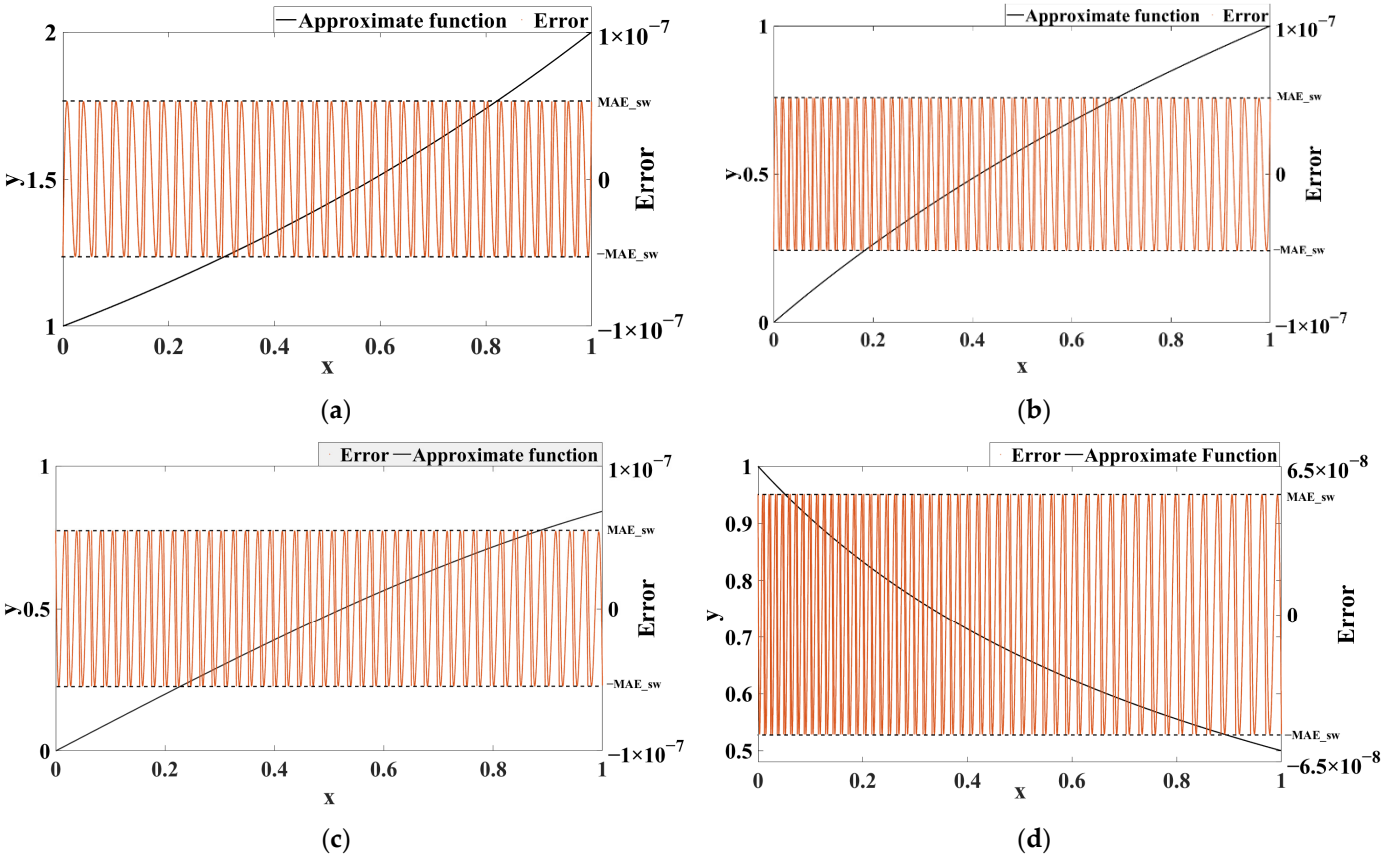

Figure 6. Error-flattening distribution diagram for different target functions in [0, 1). (**a**) 2^x; (**b**) log_2^{1+x}; (**c**) $\sin(x)$; (**d**) $1/(1+x)$.

4. Hardware Architecture and Quantizer

4.1. Hardware Architecture

The principle of the indexing of parabola coefficients is similar to that of the indexing of PWL coefficients. The conventional indexing method comprises judging segment by segment. For each judgment, a subtractor and a 2-1 MUX (multiplexer) are used in the hardware implementation. By this method, (n − 1) subtractors and (n − 1) 2-1 MUXes are cascaded to determine which segment the input x is located in. The delay increases linearly with the augment of the number of segments.

In order to solve the problem of too high a delay in the above indexing method, this paper proposes a new indexing method based on [8,9], as shown in the left side of Figure 7. For a function with n segments, the input is simultaneously compared with the starting points $S_2, S_3, \ldots, S_{n-1}$ through (n − 1) parallel subtractors. (n − 1) subtractors calculate and output sign bits $sign_i$ at the same time; then, (n − 1) symbol bits are concatenated into the MUX selection signals sign, and coefficients of corresponding segments are directly indexed. It is worth mentioning that the (n − 1) subtractors only have carry chains, which occupy much less area than ordinary subtractors. In addition, the proposed method has no encoder. The delay of this method is only one subtractor and an n − 1 MUX, which is lower than the existing method.

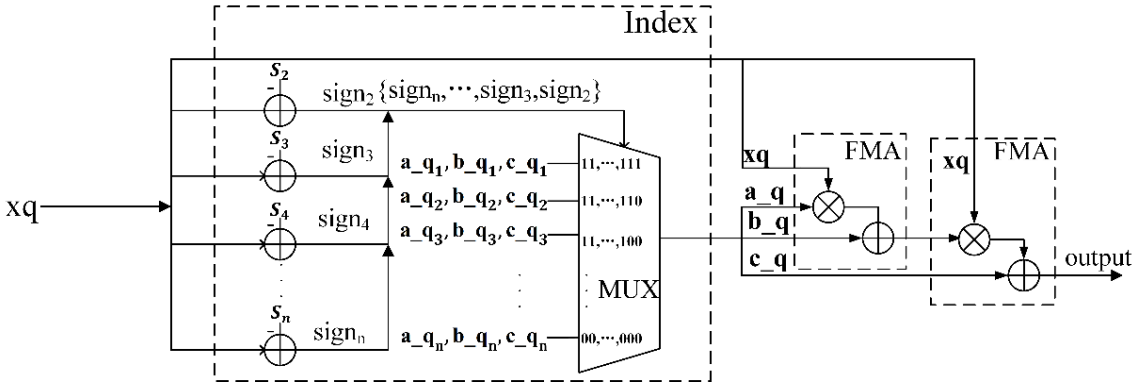

Figure 7. Hardware architecture.

Naive implementation $ax^2 + bx + c$ requires three multipliers and a carry save adder (CSA) [25]. By horner's rule: (ax + b)x + c two cascaded fused multiply-add (FMA) units are required. The FMA inserts the addend of the addition into the partial product array of the multiplication and then processes them together to reduce delay and save hardware overhead [26]. Compared to the conventional method, one multiplier is saved.

The hardware architecture is shown in Figure 7. The hardware resources include: (n − 1) subtractors (carry chains), an n − 1 MUX, and two FMA units. This architecture does not have a pipeline, in order to compare with related papers. Because the hardware architecture is a forward circuit, we can easily add registers as needed. As for the indexing of coefficients, the proposed method occupies (n − 1) subtractors (carry chains) and an n − 1 MUX, while Ref. [16] only uses an n − 1 MUX. In the calculation unit, the proposed method contains two FMAs, while Ref. [16] has one more multiplier.

4.2. A Novel Quantizer

A novel quantizer is proposed to imitate input and output behavior in hardware and quantize the related data in order to make the MAE in hardware equal to 1 ulp with a non-redundant hardware overhead. Figure 8 is a detailed description of the arithmetic circuits regarding the proposed quantizer.

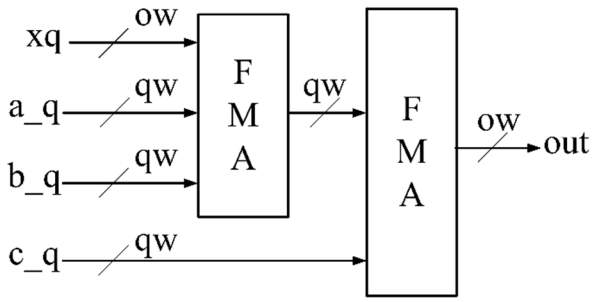

Figure 8. Arithmetic circuits regarding the proposed quantizer.

The input is discretized by the step of 2^{-ow} (ow is the width of fractional bit) in the segmenter, so the input value has been quantized in accordance with the circuits. To ensure consistency, the output fractional bit width is set equal to that of the input. Assume the approximate output in the circuits is R. Then, the maximum absolute error in circuits MAE_C is defined as

$$MAE\_C = \max |f - R|. \tag{15}$$

The quantizer aims to make MAE_C no greater than 1 ulp, which is 2^{-ow}. Firstly, the fractional bit width of coefficients a, b, and c need to be determined. In order to be accurate, they are quantized by rounding:

$$\begin{cases} a\_q = round(a \times 2^{qw}) \times 2^{-qw} \\ b\_q = round(b \times 2^{qw}) \times 2^{-qw} \\ c\_q = round(c \times 2^{qw}) \times 2^{-qw}. \end{cases} \tag{16}$$

It should be noted that, although the coefficients are all quantized to qw bits, not every bit is saved in the LUT. We only keep the required significant bits in the LUT.

The quantization of intermediate results is discussed below. The FMA combines the multiplier and adder, so we do not have to quantize the result of multiplication. For the first FMA, we only need to keep the high qw bit of the result and discard the lower bit. This operation does not require any additional hardware overhead. The quantizer can use $floor(\cdot)$ operation to completely imitate the truncation operation when the hardware is implemented. Assuming that the result of the first FMA is R1, the quantized R1 is referred to as $R1\_q$:

$$R1\_q = floor((a\_q \cdot xq + b\_q) \times 2^{qw}) \times 2^{-qw}. \tag{17}$$

The quantization of the final result remains to be discussed. If we still choose the truncation operation, this step causes an error of 1 ulp, and we cannot make the total error within 1 ulp. Thus, we round the final result to ow bits. The final output R can be expressed as:

$$R = round((R1\_q \cdot xq + c\_q) \times 2^{ow}) \times 2^{-ow}. \tag{18}$$

During hardware implementation, the round operation can be implemented by a one-bit adder. If the output is required to round to ow bits, we just need a one-bit adder to add the value in (ow+1) bit to ow bit and discard the lower bits.

Algorithm 1 depicts the quantizer in detail. The input of the quantizer includes discretized input xq, benchmark f, fractional bit width of input and output ow, a set of segment start pointers $S = S_1, S_2, S_3, \ldots, S_n$, a set of segment end pointers $E = E_1, E_2, E_3, \ldots, E_n$, the number of segments n, and the number of sample points for the objective function NUM (refer to Equation (2)). It should be noted that the end pointers set E does not appear in the hardware implementation, because the location of the segment can be determined only by the start pointers. This concept is introduced here to simplify the description of the algorithm.

Algorithm 1: A novel quantizer

Input Discretized input xq,
benchmark f,
fractional bit width of input and output ow,
set of segment start pointers $S = \{S_1, S_2, S_3, \ldots, S_n\}$,
set of segment end pointers $E = \{E_1, E_2, E_3, \ldots, E_n\}$,
number of segments n,
number of sample points for the objective function NUM;

output Fractional bit width of coefficients and intermediate results qw,
MAE in circuits MAE_C;

Initialization

1: MAE_C=2 ulp; % initial MAE_C
2: $gw = -1$; % initial guard bit width

Begin

5: **while** (MAE_C > 1 ulp) % fail to search guard bit
6: gw = gw +1; % update gw
7: $qw = ow + gw$; % update qw
8: $a\_q = \text{round}\,(a \times 2^{qw}) \times 2^{-qw}$; % quantize a
9: $b\_q = \text{round}\,(b \times 2^{qw}) \times 2^{-qw}$; % quantize b
10: $c\_q = \text{round}\,(c \times 2^{qw}) \times 2^{-qw}$; % quantize c
11: **for** i=1:n **do**
12: $R1\_q(S_i{:}E_i) = \text{floor}((a\_q(i) \times xq(S_i{:}E_i) + b\_q(i)) \times 2^{qw}) \times 2^{-qw}$;
13: $R(S_i{:}E_i) = \text{round}((R1\_q(S_i{:}E_i) \times xq(S_i{:}E_i) + c\_q(i)) \times 2^{ow}) \times 2^{-ow}$
% calculate result in circuits
17: **end for**
18: $MAE\_C = \max(|f - R|)$; % MAE in circuits
19: **end while**

end

The output of the quantizer includes the fractional width of coefficients and intermediate results qw, and the MAE in circuits MAE_C. The main target of the quantizer is to find the smallest qw to ensure that the MAE_C is less than or equal to 1 ulp. Due to error propagation, setting qw equal to ow may not meet the circuit's error requirements, and additional guard bits gw is required. Therefore, qw can be expressed as $qw = ow + gw$. The main purpose of the quantizer is to continuously increase guard bits gw and calculate MAE_C until it is less than or equal to 1 ulp.

In order to explore the relationship between the error requirements in software and hardware, we introduce a trade-off factor called TF, which is defined as

$$TF = \frac{MAE\_sw}{1\ ulp}. \tag{19}$$

The MAE_sw in the segmenter can be determined by adjusting TF:

$$MAE\_sw = TF \times 1\ ulp, \qquad TF < 1. \tag{20}$$

The error of the circuit consists of three aspects: the algorithm error, the coefficient and intermediate result quantization error, and the rounding error. The algorithm error is caused by the gap between the approximate quadratic function and the objective function. The MAE of the algorithm error is expressed as MAE_sw, which is analyzed in the segmenter. The coefficient and intermediate result quantization error is written as MAE_quantized. The rounding error is caused by the rounding operation of the output, and its MAE is recorded as MAE_round. In order not to lose generality, we consider the most pessimistic case—that is, the total error is the algebraic sum of the above three errors. The total error is less than or equal to 1 ulp. Thus:

$$\text{MAE\_sw} + \text{MAE\_quantized} + \text{MAE\_round} \leq 1 \text{ ulp}. \quad (21)$$

Because of MAE_round is 0.5 ulp, we can get the following relationship:

$$\text{MAE\_sw} < 0.5 \text{ ulp}. \quad (22)$$

Therefore, TF is less than 0.5. We can change TF between 0 and 0.5. With the ulp unchanged, the smaller TF is, the smaller the segmenter's error requirement MAE_sw is, which may lead to too many unnecessary segments. The larger TF is, the bigger the MAE_sw is. Although the quantizer can obtain the desired effect by increasing the guard bit width, an excessive bit width will lead to an unnecessary increase in area. Therefore, designers have the flexibility to choose TF for better trade-offs.

In general, TF is chosen by two steps. Firstly, candidates are chosen by observing the error distribution of the last segment. If the maximum absolute error of the last segment is not equal to MAE_sw, as shown in Figure 9a, the minor reduction in TF will only cause the error to increase but will not increase the number of segments. Smaller TF will strive for more space for quantization error, which is a pure optimization. Therefore, TF is not the candidate. When the maximum absolute error in the last segment is equal to MAE_sw, as shown in Figure 9b, the reduction in TF will cause an increase in the number of segments, which is undesirable. Thus, TF is the candidate. For the sinusoidal function $\sin(x), x \in [0,1)$ with the ow of 23, the candidate TF and related parameters are shown in Table 2. Secondly, the optimal solution with the minimum LUT size, which is calculated through the number of segments and bit width, is chosen from the candidates.

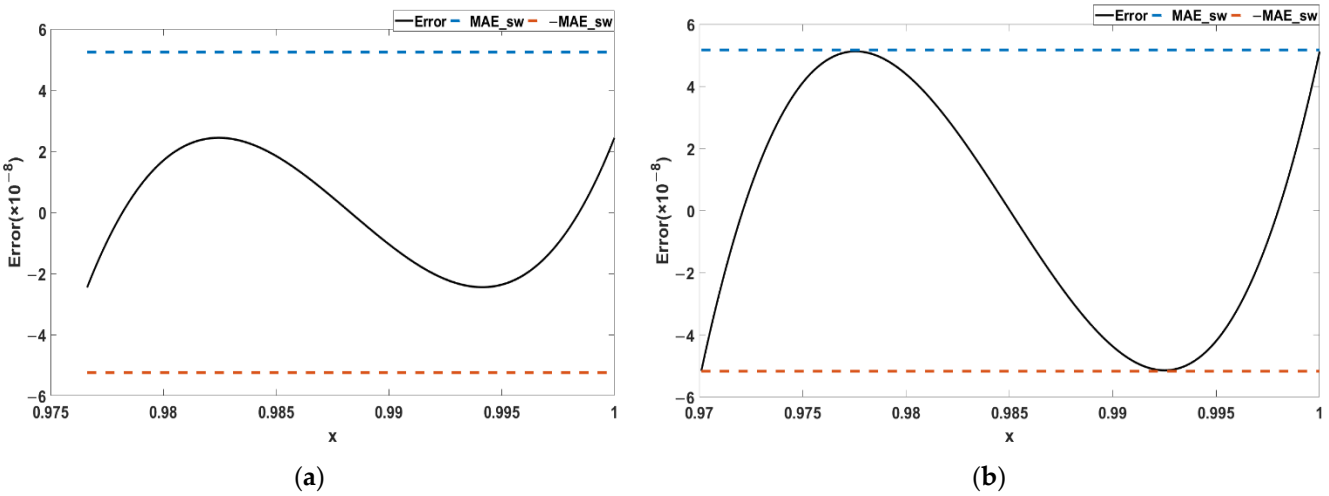

Figure 9. The impact of TF on the error distribution of the last segment. (**a**) The maximum absolute error of the last segment is not equal to MAE_sw; (**b**) the maximum absolute error in the last segment is equal to MAE_sw.

Table 2. Impact of TF.

Function	TF	Number of Segments	q	LUT(bit)
$y = \sin(x)$ $x \in [0,1)$	0.493	42	31	3096
	0.459	43	28	3612
	0.428	44	28	3696
	0.401	45	28	3780
	0.375	46	27	3726

5. Hardware Implementation and Comparison

5.1. Simulation on CPU

We perform the proposed algorithm of the segmenter and quantizer in MATLAB and test the computing time under different functions and different accuracies. Table 3 reports the CPU time running on an AMD Ryzen 5 Six-Core Processor with 16 GB RAM. The CPU time of different functions with the same precision is slightly different because of the different properties of the functions. The CPU time with different precision varies greatly, from tens of milliseconds to hundreds of seconds, but they are all acceptable. In the worst case of the test, for the cosine function with the accuracy of 2^{-27}, the total required CPU time is only about six minutes.

Table 3. Simulation time on CPU.

Function	Accuracy	Segmenter CPU Time(s)	Quantizer CPU Times(s)
$y = \cos(\frac{\pi}{2} \times x)$	2^{-16}	0.02	0.03
	2^{-23}	8.28	3.64
	2^{-25}	52.41	18.97
	2^{-27}	318.58	92.00
$y = (\frac{1}{\sqrt{1+x}})$	2^{-16}	0.02	0.03
	2^{-23}	8.44	6.68
	2^{-25}	48.85	31.68
	2^{-27}	275.01	126.20
$y = \ln(1 + e^{x})$	2^{-16}	0.02	0.03
	2^{-23}	4.50	5.01
	2^{-25}	23.47	19.75
	2^{-27}	127.18	66.56

5.2. Comparison of LUT Size

To highlight the advantages of our method with regard to the size of LUT, we compare it with many existing methods, including the most widely accepted uniform segmentation method [15], the non-uniform segmentation method [22], the most advanced method using ILP [16], and other classic methods [11,12,27].

Table 4 shows in detail the number of segments, coefficient bit width, and LUT size of various methods under different functions when the accuracy is 2^{-16} and 2^{-24}, respectively. The size of LUTs compared to [11,12,16,27] for various functions and various accuracies is reported in Table 5. It can be seen that the proposed method has the smallest LUT size and number of segments for all functions and all precisions tested. As for the bit width of coefficients, in many cases, our method is slightly inferior, which makes the arithmetic unit larger. The LUT size is a trade-off with the bit width of the arithmetic unit. The proposed method is excellent in the number of segments and LUT size, leaving little space for bit width optimization. The proposed method is suitable for applications where the size of LUTs has a great impact on performance.

Table 4. Comparison in segments, LUT size, and coefficient width.

Function	Accuracy	Method	Number of Segments	c bit	b bit	a bit	LUT Size(bit)	Normal of LUT Size
$y = \frac{1}{x}$	2^{-16}	Uniform [15]	16	22	20	18	960	1.56
		Non-uniform[22]	13	22	20	18	780	1.27
		This paper	11	17	19	20	616	1
	2^{-24}	Uniform[15]	128	30	25	20	9600	1.75
		Non-uniform[22]	91	30	25	20	6825	1.25
		This paper	66	26	28	29	5478	1
$y = log_2(x)$	2^{-16}	Uniform[15]	16	22	21	18	976	1.84
		ILP[16]	16	19	9	10	608	1.15
		Non-uniform[22]	13	22	21	18	739	1.39
		This paper	9	18	20	21	531	1
	2^{-24}	Uniform[15]	128	30	26	20	9728	2.06
		ILP[16]	128	26	15	10	6556	1.39
		Non-uniform[22]	80	30	26	20	6080	1.29
		This paper	57	26	28	29	4731	1
$y = 2^x$	2^{-16}	Uniform[15]	16	24	18	17	944	2.27
		Non-uniform[22]	15	24	18	17	885	2.13
		This paper	8	16	19	17	416	1
	2^{-24}	Uniform[15]	64	32	23	15	4480	1.17
		Non-uniform[22]	64	32	23	15	4480	1.17
		This paper	46	26	29	28	3818	1
$y = 2^{-x}$	2^{-16}	Uniform[15]	8	20	17	13	400	1.33
		Non-uniform[22]	8	20	17	13	400	1.33
		This paper	6	15	18	17	300	1
	2^{-24}	Uniform[15]	64	31	23	18	4608	1.60
		Non-uniform[22]	64	31	23	18	4608	1.60
		This paper	38	24	26	26	2888	1

Table 5. Comparison in LUT size.

Function	Accuracy	Method	LUT Size (bit)	Normal of LUT Size
$y = \ln(1 + x)$	2^{-18}	[12]	1664	1.60
		[16]	1312	1.26
		This paper	1040	1
	2^{-24}	[12]	7808	1.85
		[16]	6912	1.64
		This paper	4212	1
$y = 1/(1 + x)$	2^{-15}	[11]	528	1.18
		[16]	560	1.25
		[27]	672	1.50
		This paper	448	1
	2^{-19}	[11]	2432	1.64
		[16]	1696	1.15
		[27]	2880	1.95
		This paper	1480	1
	2^{-23}	[11]	6144	1.48
		[16]	6656	1.60
		[27]	7040	1.69
		This paper	4160	1

5.3. Implementations and Comparisons in 90 nm CMOS

The proposed hardware architecture is coded in Verilog HDL and implemented in 90 nm technology. We synthesize the circuits by Synopsys Design Compiler and present the implementation results in Table 6.

Ref. [10] and [16] reported several implementation results in 90 nm technology of 23-bit rounded interpolators using: the classic Chebyshev quadratic interpolation method [13], minimax degree-two polynomial approximation [15], the two-level function evaluation approach proposed in [10], and the ILP method targeted at minimizing coefficients' bit width in [16]. For the fairness of comparison, we implemented the same set of functions, with the same accuracy and the same technology, using our technique. These functions cover the common function types in hardware implementation, which are representative. In addition, the proposed method has obvious advantages in middle-to-high-precision applications. These functions usually require medium to high precision when implemented in hardware. We also designed a multi-function unit that can compute several arithmetic functions in the same hardware as in [10].

We test the LUT and address generation area of the proposed method for several single functions and compare it to the LUT area (without address generation area) in [10,13,15]. Obviously, the proposed method is better. This is consistent with the previous analysis. Ref. [16] did not provide the result of the LUT area. Moreover, the method proposed in this paper lacks details, and so we cannot reproduce it. However, the number of segments of uniform segmentation method is almost the same, and so the proportion of LUT area in the total area is almost the same. We take the average value of the proportion of LUT area in [13] and [15] as the proportion of LUT area in [16]. We can obtain the estimated value of LUT area in [16]. As shown in Table 6, the proposed method has a smaller LUT and address generation area than [16] for all cases in the table.

For the implementation of single functions, the synthesis results show that the proposed method is superior to the classical method in area and delay [10,13,15]. For the functions $y = \frac{1}{\sqrt{x}}$ and $y = \sqrt{x}$, the proposed method has advantages over ILP [16], which is the state-of-the-art. Compared with ILP, the proposed method has a smaller area and a higher delay for the functions $y = \log_2(x)$, $y = 2^x$ and $y = (1/x)$. Meanwhile, for the function $y = \sin(x)$, the area of the proposed method has weak disadvantage over ILP in area and delay. To provide a fair comparison, we use $\text{area} \times \text{delay}$ as a new metric of cost. As shown in Table 6, the proposed method has a lower cost than [10,13,15]. For the function, $y = \log_2(x)$, $y = \frac{1}{\sqrt{x}}$, $y = \frac{1}{x}$, and $y = \sqrt{x}$, the proposed method costs less than in [16].

For the multi-function unit, each function needs an individual LUT, but the calculation unit can be reused. The hardware architecture of the proposed multi-function unit is shown in Figure 10. For the methods in this paper and references, the index units of all functions are accumulated, while the calculation units are reused, although the index units and calculation units in this paper and references are different. The method proposed has apparent advantages because the LUT is very small. The synthesis results support this view. Compared with the methods in [13,15,16], the proposed method saves 58.67%, 42.86%, 35.06% of the area and 35.48%, 66.23%, and 50.75% of the delay, respectively.

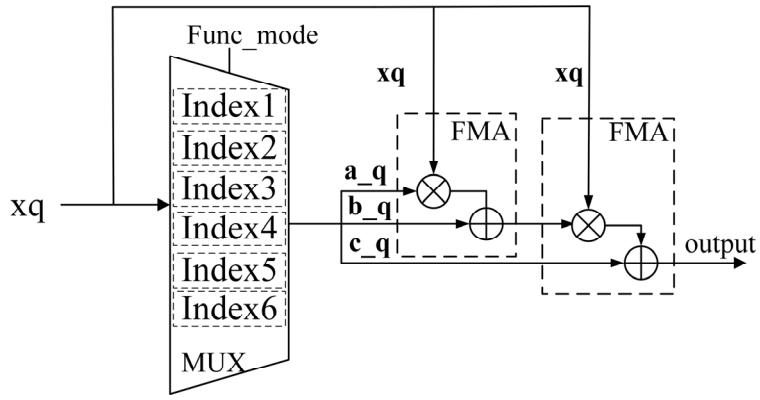

Figure 10. Hardware architecture for multi-function unit.

Table 6. Comparison of synthesized results.

Function	Method	LUT & Address Generation Area [μm²]	Total Area [μm²]	Normal Area	Delay [ns]	Normal Delay	Area × Delay [μm² × ns]
$y = \frac{1}{x}$	Cheyshev [10,13]	7228	28,886	1.53	9.28	1.39	268,062
	Minimax [10,15]	6945	28,428	1.50	9.71	1.46	276,036
	Two-level [10]	5595	24,023	1.27	12.69	1.91	304,852
	ILP [16]	5325 [1]	21,560	1.14	6.36	0.95	137,122
	This paper	2804	18,908	1	6.66	1	125,927
$y = 2^x$	Cheyshev [10,13]	4270	29,063	1.62	9.79	1.48	284,527
	Minimax [10,15]	3880	26,635	1.49	9.8	1.48	261,023
	Two-level [10]	5452	18,913	1.06	12.16	1.84	229,982
	ILP [16]	2273 [1]	18,963	1.06	5.79	0.88	109,796
	This paper	2159	17,905	1	6.60	1	118,173
$y = \frac{1}{x}$	Cheyshev [10,13]	6859	28,177	1.46	9.75	1.46	274,726
	Minimax [10,15]	6712	27,855	1.44	9.71	1.45	270,472
	Two-level [10]	5564	24,137	1.25	12.63	1.89	304,850
	ILP [16]	5377 [1]	22,243	1.15	6.92	1.03	153,922
	This paper	3063	19,310	1	6.69	1	129,184
$y = \sqrt{x}$	Cheyshev [10,13]	10,230	31,700	2.00	9.36	1.38	296,712
	Minimax [10,15]	3627	26,036	1.64	9.8	1.45	255,153
	Two-level [10]	4437	16,043	1.01	12.03	1.77	192,997
	ILP [16]	4072 [1]	17,618	1.11	6.35	0.94	111,874
	This paper	1456	15,888	1	6.78	1	107,721
$y = \frac{1}{\sqrt{x}}$	Cheyshev [10,13]	18,453	39,737	2.29	9.45	1.50	375,515
	Minimax [10][15]	6535	29,678	1.71	9.71	1.54	288,173
	Two-level [10]	5388	22,317	1.29	12.83	2.03	286,327
	ILP [16]	6315[1]	18,460	1.07	6.89	1.09	127,189
	This paper	2254	17,332	1	6.31	1	109,365
$y = \sin(x)$	Cheyshev [10,13]	4015	28,144	1.48	9.52	1.42	267,931
	Minimax [10,15]	3795	26,400	1.39	9.67	1.44	255,288
	Two-level [10]	5496	22,890	1.21	13.08	1.95	299,401
	ILP [16]	2684 [1]	18,743	0.99	6.28	0.94	117,706
	This paper	2566	18,993	1	6.71	1	127,443
Multi-function	Cheyshev [10,13]	51,052	77,864	2.42	11.50	1.55	895,436
	Minimax [10,15]	31,496	56,224	1.75	11.22	1.51	630,833
	Two-level [10]	16,211	49,548	1.54	15.07	2.03	746,688
	ILP [16]	26,546 [1]	43,668 [1]	1.36 [1]	8.03 [1]	1.08 [1]	350,654 [1]
	This paper	13,757	32,182	1	7.41	1	238,469

[1] Estimated result.

Ref [16] did not provide the result of the multi-function unit. As mentioned above, the method proposed in this paper lacks the details necessary for us to reproduce it, so we make a reasonable estimate for it. As for uniform segmentation, the LUT area for the multi-function unit equals the sum of the LUT area of each function, which can be confirmed by the data in [13,15]. Therefore, we can obtain the estimated LUT area of multi-function unit for [16]. According to the previous analysis, we take the average value of the proportion of LUT area in [13,15] as the proportion of LUT area in [16] and calculate the total area of multi-function unit for [15]. The delay of the multi-function unit is related to the maximum delay of all functions. We calculate that of the multi-function unit as being 1.14 times and 1.17 times the maximum delay of all functions in [13,15], respectively. We estimate that the delay of the multi-function unit is 1.155 times the maximum delay of all functions in [16], and obtain the estimated results. Our method saves 48.18% of the area and 7.72% of the delay when compared with [16]. Thus, for the multi-function unit, our method has better performance than the state of the art.

6. Conclusions

This paper presents an area-efficient piecewise parabolic approximate computation method for all unary functions with finite codomain, in which the segmenter is first responsible for performing an error-flattened segmentation process, and the quantizer is then accountable for quantizing the data width to guarantee a non-redundant fixed-point hardware architecture with the MAE of 1 ulp.

For six common functions, the proposed method reduces the area with similar delay. A multi-function unit is implemented by the proposed method for 90 nm technology. Compared to the method in [13], the proposed method reduces the area by 58.67% and the delay by 35.48%. In addition, our method saves 42.86% of the area and 66.23% of the delay when compared with [15]. In addition, the proposed method cuts 35.06% of the area and 50.75% of the delay compared with [10]. The proposed method saves 48.18% of the area and 7.72% of the delay compared with [16].

Author Contributions: Conceptualization, M.A. and Y.L.; methodology, M.A.; software, M.A. and Y.L.; validation, M.Z. and M.A.; investigation, M.Z. and H.D.; data curation, M.A. and Y.L.; writing—original draft preparation, M.A.; writing—review and editing, Y.W., Z.W., and C.P.; supervision, H.P.; project administration, H.P.; funding acquisition, H.P. and C.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by the National Nature Science Foundation of China via grants 61376075 and 41412020201; in part by the Natural Science Foundation of Jiangsu Province of China via grants BK20211149; in part by the key Research and Development Program of Jiangsu Province under Grant No. BE2015153; and in part by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD). The APC was funded by the National Nature Science Foundation of China under Grant No. 61376075.

Acknowledgments: This work was supported in part by the National Nature Science Foundation of China via grants 61376075 and 41412020201; in part by the Natural Science Foundation of Jiangsu Province of China via grants BK20211149; in part by the key Research and Development Program of Jiangsu Province under Grant No. BE2015153; and in part by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Oberman, S.F.; Siu, M.Y. A high-performance area-efficient multifunction interpolator. In Proceedings of the 17th IEEE Symposium on Computer Arithmetic (ARITH'05), Cape Cod, MA, USA,27–29 June 2005; pp. 272–279.
2. Ng, S.W. A variable dimension Newton method. *IEEE Int. Symp. Circuits Syst.* **1994**, *6*, 129–132.
3. Oberman, S.F. Floating point division and square root algorithms and implementation in the amd-k7TM microprocessor. In Proceedings of the IEEE Symposium on Computer Arithmetic (Cat. No.99CB36336), Adelaide, Australia, 14–16 April 1999; pp. 106–115.

4. Volder, J.E. The CORDIC trigonometric computing technique. In *Proceedings of the IRE Transactions on Electronic Computers*; IEEE: New York, NY, USA, 1959; pp. 330–334.
5. Luo, Y.; Wang, Y.; Ha, Y.; Wang, Z.; Chen, S.; Pan, H. Generalized hyperbolic CORDIC and its logarithmic and exponential computation with arbitrary fixed base. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 2156–2169.
6. Das Sarma, D.; Matula, D.W. Faithful bipartite ROM reciprocal tables. In Proceedings of the 12th Symposium on Computer Arithmetic, Bath, UK, 19–21 July 1995; pp. 17–28.
7. Muller, J.-M. A few results on table-based methods. In *Developments in Reliable Computing*; Springer: Dordrecht, Germany, 1999; pp. 279–288.
8. Sun, H.; Luo, Y.; Ha, Y.; Shi, Y.; Gao, Y.; Shen, Q.; Pan, H. A universal method of linear approximation with controllable error for the efficient implementation of transcendental functions. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2020**, *67*, 177–188.
9. Dong, H.; Wang, M.; Luo, Y.; Zheng, M.; An, M.; Ha, Y.; Pan, H. PLAC: Piecewise linear approximation computation for all nonlinear unary functions. *IEEE Trans. VLSI* **2020**, *28*, 2014–2027.
10. Hsiao, S.; Ko, H.; Wen, C. Two-level hardware function evaluation based on correction of normalized piecewise difference functions. *IEEE Trans. Circuits Syst. II Express Briefs* **2012**, *59*, 292–296.
11. Strollo, A.G.M.; De Caro, D.; Petra, N. Elementary functions hardware implementation using constrained piecewise-polynomial approximations. *IEEE Trans. Comput.* **2011**, *60*, 418–432.
12. Lee, D.; Cheung, R.; Luk, W.; Villasenor, J. Hardware implementation trade-offs of polynomial approximations and interpolations. *IEEE Trans. Comput.* **2008**, *57*, 686–701.
13. Schulte, M.J.; Swartzlander, E.E. Hardware designs for exactly rounded elementary functions. *IEEE Trans. Comput.* **1994**, *43*, 964–973.
14. Muller, J.-M. Elementary functions. In *Algorithms and Implementation*, 3rd ed.; Springer: New York, NY, USA, 2016.
15. Pineiro, J.; Oberman, S.F.; Muller, J.; Bruguera, J.D. High-speed function approximation using a minimax quadratic interpolator. *IEEE Trans. Comput.* **2005**, *54*, 304–318.
16. De Caro, D.; Napoli, E.; Esposito, D.; Castellano, G.; Petra, N.; Strollo, A.G.M. Minimizing coefficients wordlength for piecewisepolynomial hardware function evaluation with exact or faithful rounding. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2017**, *64*, 1187–1200.
17. Ellaithy, D.M.; El-Moursy, M.A.; Zaki, A.; Zekry, A. Dual-channel multiplier for piecewise-polynomial function evaluation for low-power 3-D graphics. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, 27, 790–798.
18. Lee, D.; Cheung, R.C.C.; Luk, W.; Villasenor, J.D. Hierarchical segmentation for hardware function evaluation. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2009**, *17*, 103–116.
19. Lee, D.; Luk, W.; Villasenor, J.; Cheung, P.Y.K. Hierarchical segmentation schemes for function evaluation. In Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT) (IEEE Cat. No.03EX798), Tokyo, Japan, 9–11 December 2003; pp. 92–99.
20. Ge, L.; Chen, S.; Yoshimura, T. Automatic implementation of arithmetic functions in high-level synthesis. In Proceedings of the 9th International Conference on Solid-State and Integrated-Circuit Technology, Beijing, China, 20–23 October 2008; pp. 2349–2352.
21. Sasao, T.; Nagayama, S.; Butler, J.T. Numerical function generators using LUT cascades. *IEEE Trans. Comput.* **2007**, *56*, 826–838.
22. Ko, H.; Hsiao, S.; Huang, W. A new non-uniform segmentation and addressing remapping strategy for hardware-oriented function evaluators based on polynomial approximation. In Proceedings of the IEEE International Symposium on Circuits and Systems, Paris, France, 30 May–2 June 2010; pp. 4153–4156.
23. Hsiao, S.; Ko, H.; Tseng, Y.; Huang, W.; Lin, S.; Wen, C. Design of hardware function evaluators using low-overhead nonuniform segmentation with address remapping. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2013**, *21*, 875–886.
24. Cao, J.; Wei, B.W.Y.; High-performance hardware for function generation. In Proceedings of the 13th IEEE Symposium on Computer Arithmetic, Asilomar, CA, USA, 6–9 July 1997; pp. 184–188.
25. Kuang, S.; Wang, J.; Chang, K.; Hsu, H. Energy-efficient high throughput Montgomery modular multipliers for RSA cryptosystems. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2013**, *21*, 1999–2009.
26. Zhang, H.; Chen, D.; Ko, S. Efficient multiple-precision floating point fused multiply-add with mixed-precision support. *IEEE Trans. Comput.* **2019**, *68*, 1035–1048.
27. Walters, E.G.; Schulte, M.J. Efficient function approximation using truncated multipliers and squarers. In Proceedings of the 17th IEEE Symp. Computer Arithmetic (ARITH'05), Cape Cod, MA, USA, 27–29 June 2005; pp. 232–239.