A Division-Free Algorithm for Fixed-Point Power Exponential Function in Embedded System

Chung-Hsien Chang[1], Shi-Huang Chen[2], Bo-Wei Chen[1†], Jia-Ching Wang[3], and Jhing-Fa Wang[1,4]

[1]Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, Email: n28981387@mail.ncku.edu.tw;
[2]Department of Computer Science Information Engineering, Shu-Te University, Kaohsiung, Taiwan. Email: shchen@stu.edu.tw
[3]Department of Computer Science and Information Engineering, National Central University, Jhongli City, Taiwan
[4]Department of Digital Multimedia Design, Tajen University, Pingtung, Taiwan

***Abstract*—This work presents a division-free algorithm for fixed-point power exponential function (PEF) using Newton's method. Such a mechanism can improve the computational speed of PEF and is suitable for low-cost embedded systems without floating-point units (FPU). To achieve the goal, this work develops a fast square method to effectively describe a PEF in the form of multiplicative representation. Such representation can be separated into integer and fraction parts. For computing the base term of fraction part in fast square method, a division-free Newton's method is proposed in this paper. The proposed one utilizes two-stage iterations to modify the conventional solving strategy to reduce iteration times when the exponential term is positive. The experimental results show that the proposed algorithm can reduce the execution period about 1.8 times than the baseline one. Additionally, the performance of the proposed algorithm can reach five times higher than that of the system using a floating architecture. The computational precision of the proposed algorithm is also closed to that of the algorithm using floating operations.**

Keywords—Power exponential function, fixed-point mathematical function, Newton's method

I. INTRODUCTION

The power exponential function (PEF) is a general and useful mathematical function. It has been widely used in graphic processing, engineering computing, and statistics computing. The PEF is also applied to efficiently calculate some special mathematic operations, such as forward/inverse square root, division or forward/inverse 2-based norm functions. Basically, the computing procedure of PEF in the standard C library is implemented by the floating mathematic operations using floating-point unit (FPU). It will become a bottleneck during the development of the software based on the low-end embedded system that comes without FPU. It is because that the cross compiler for the low-end embedded system will utilize the integer instructions to simulate the behavior of floating point operations. It will result in encumbrance for software computing times when multiple mathematical functions are called simultaneously.

Several papers [1], [2] have proposed related hardware or algorithm for these useful mathematical operations with floating-point data structure to speed-up the software routine. For the fixed-point data representation in forward/inverse square root and forward/inverse logarithm problems, various methods are developed in [3]−[6]. In this paper, a fixed-point PEF that only uses the integer addition/shift/multiplication operations is proposed to speed up the PEF based on low-cost embedded systems.

The rest of this paper is organized as follows. In Section II the preliminaries include the fixed-point representation and the Newton's method are given. Sections III and IV describe the original algorithm and the proposed algorithm for fixed-point power exponential function. The comparative experimental results of the proposed algorithm, original algorithm, and the baseline one are shown in Section V. Finally, Section VI concludes this paper.

II. PRELIMINARIES

A. The Representation of Fixed-Point Data Structure

The data format of a fixed-point number is included fraction and integer part which can be described in Fig. 1

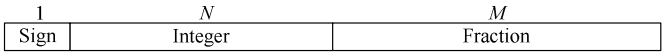

Fig. 1. The data format for a fixed-point number

where N is the bits number of the integer part, and M is the bits number of fraction part.

Compared with the floating data structure, such a data format has fixed precision with more narrow data range and also has good precision with the same bits length. It is suitable for representing variable sequence with a stable range and average probability. A fixed-point number can be also described using a bit-wise summation representation. That is

$$A = \alpha \sum_{n=-M}^{N-1} \beta_n 2^n,\ \alpha = 1 \text{ or } -1,\ \beta_n = 1 \text{ or } 0 \qquad (1)$$

where α is the sign bits, and β_n represents the n^{th} bits in the integer or the fraction part.

B. Newton's method

The Newton's method is a well-known method to find the root of a non-linear function using an approximating strategy. Its iteration formula is described as (2).

978-1-4673-5936-8/13/$31.00 ©2013 IEEE

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

where n is the iteration times, f is the targeted non-linear function, and x is solution for the target function. The iteration times and convergence criterion of Newton's method are decided by the initial value, and end the iteration by the residual threshold ε_{th}. The Newton's iterative procedure (NIP) can be described by using the simple pseudo-code as follow

TABLE I. THE NEWTON'S ITERATIVE PROCEDURE

```
procedure NIP(f)
n=1
Compute the x_n using iteration formula with targeted f
while(v < ε_th) do
  Compute the x_{n+1} using iteration formula with targeted f
  v = |x_{n+1} − x_n|
  x_n = x_{n+1}
  n=n+1
end do
return x_n
```

This method can also effectively solve some special fixed-point mathematical operation problems (e.g. division, inverse square root) only by the use of addition/multiplication/shift operations. This issue will be further discussed in Section III.

C. The Definition of PEF Problem by Multiplicative Representation

A fixed-point PEF with two real numbers input can be described as (3) according to (1), where the fixed-point PEF data representation can be written by using a multiplicative representation following equation. That is,

$$\begin{aligned} B^A &= B^{\alpha\sum_{n=-M}^{N-1}\beta_n 2^n} = \left(B^{\alpha\sum_{n=-M}^{-1}\beta_n 2^n}\right)\left(B^{\alpha\sum_{n=0}^{N-1}\beta_n 2^n}\right) \\ &= \left(\prod_{n=-M}^{-1} B^{\alpha\beta_n 2^n}\right)\left(\prod_{n=0}^{N-1} B^{\alpha\beta_n 2^n}\right) = B^{A_F} B^{A_I} = C \end{aligned} \tag{3}$$

where B^{A_F} is the fraction part, B^{A_I} is the integer part of the base number B, and the C is the answer. The relationship between each two adjacent terms can be described as (4),

$$B^{\alpha 2^{-k+1}} = B^{\alpha 2^{-k}} B^{\alpha 2^{-k}}, -M \le k \le N \tag{4}$$

It follows from (4) that the term of k+1 order can be calculated by the term of k order. Therefore, if the first terms of B^{A_I} and B^{A_F} are known, the B^{A_I} and B^{A_F} can be computed by this relationship and it can use a fast square method (FSM) [7] to reduce the computing complexity from $O(n)$ to $O(\log n)$. The procedure of FSM is shown in TABLE II. In addition, TABLE III summarizes the first term of B_I, and B_F with different α

TABLE II. THE PROCEDURE OF FAST SQUARE METHOD

```
procedure FSM(x,y)
t=1
while(y!=0) do
  if((y&1)!=0) do
    t=t*x
  end do
  y=y>>1
  x=x^2
end do
return t
```

TABLE III. THE FIRST TERMS IN B_I AND B_F

α	Integer/Fraction part	First term
-1 (A<0)	B_I	B^{-1}
	B_F	$B^{-2^{-M}}$
1 (A≥0)	B_I	B
	B_F	$B^{2^{-M}}$

The issue discussed above is focused on how to calculate the first term of the integer and fraction parts, (i.e. B_I, and B_F in (3)).

III. THE ORIGINAL ALGORITHM

Due to the parameter B in TABLE I is not necessary to be calculated, therefore, the algorithm of fixed-point PEF aims to solve these three terms, i.e., B^{-1}, $B^{-2^{-M}}$ and $B^{2^{-M}}$ by using the Newton's method. The term B^{-1} can be calculated by the following iteration equation.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^{-1} - B}{-x_n^{-2}} = x_n\left(2 - Bx_n\right) \tag{5}$$

The term B^{-1} is computed with NIP using only two multiplications and one addition in one iteration. Therefore, the terms $B^{-2^{-M}}$ and $B^{2^{-M}}$ also can be obtained with similar strategy given in (6) and (7), respectively

$$x_{n+1} = x_n - \frac{x_n^{-2^M} - B}{-2^M x_n^{-(2^M+1)}} = x_n + \left(\left(x_n - Bx_n^{2^M+1}\right) >> M\right) \tag{6}$$

$$x_{n+1} = x_n - \frac{x_n^{2^M} - B}{2^M x_n^{2^M-1}} = x_n - \left(\left(x_n - \frac{B}{x_n^{2^M-1}}\right) >> M\right) \tag{7}$$

where the terms $x_n^{2^M+1}$ and $x_n^{2^M-1}$ in (6) and (7) can be obtained by using FSM for fast computation. For the purpose of obtaining the $B^{2^{-M}}$, the term $B / x_n^{2^M-1}$ in (6) still needs a division operation to obtain the $1/x_n$ using (5) and hence FSM operation. According to the equations described above, the computation times of additions, multiplications, and shift operations in NIP of (5), (6), and (7) are shown in TABLE IV.

TABLE IV. THE COMPUTATION TIMES OF EACH ITERATIONS USING NEWTON'S METHOD

Equation	Additions	Multiplications	Division solved by (6)	Shift operations
(5)	1	2	0	0
(6)	2	2*M*+1	0	1
(7)	2	2*M*+1	1	1

The procedure of the original fixed-point PEF is shown in TABLE V, where the FT_{FRA} and FT_{INT} are first terms of integer part B_I and fraction part B_F, respectively. The B_I and the B_F will be obtained by using FSM with low computing complexity. Finally, the B_I multiplied by B_F is the return value of original function. Nevertheless, if the sign bits α is positive, the original one is truly needed more times to calculate $B^{2^{-M}}$. Such a strategy will lead to a high-cost computation load in the iteration procedure.

TABLE V. THE ORIGINAL ALGORITHM FOR FIXED-POINT PEF

procedure *OFPEF* (α, A, B)
Compute the A_I and A_F by Masking A
if (α=−1) do
$FT_{INT} = NIP\left(B^{-1}\right)$
$FT_{FRA} = NIP\left(B^{-2^{-M}}\right)$
else
$FT_{INT} = B$
$FT_{FRA} = NIP\left(B^{2^{-M}}\right)$
end do
$B_I = FSM\left(FT_{INT}, A_I\right)$
$B_F = FSM\left(FT_{INT}, A_F\right)$
return $\boldsymbol{B_I \times B_F}$

IV. THE PROPOSED ALGORITHM

To reduce the computation time of calculating the first term of fraction part when α=1 in the original fixed-point PEF, this paper presents an improved algorithm using Newton's method with different problem definition to solve the question that previously introduced. The simple idea in the proposed algorithm is to reuse the (5) and (6) to change the problem definition from $B^{2^{-M}}$ to $\left(B^{-2^{-M}}\right)^{-1}$. The procedure of the improved algorithm is shown in TABLE VI.

where the variable V is the value of the $B^{-2^{-M}}$. The proposed algorithm can prevent the iterative division operation in (7). Such a simple strategy effectively reduces the iteration times needed in the original ones. It only uses two-stage iterations to reduce the iteration times. The computational complexity of the proposed algorithm for computing the FT_{INT} will be straightly reduced from $O(n^2)$ to $O(n)$ if the sign bit α is positive. On the other hand, the proposed algorithm only needs to reuse the same procedure code of (5) and (6)

TABLE VI. THE IMPROVED ALGORITHM FOR FIXED-POINT PEF

procedure *IFPEF* (α, A, B)
Compute the A_I and A_F by Masking A
if (α= −1) do
$FT_{INT} = NIP\left(B^{-1}\right)$
$FT_{FRA} = NIP\left(B^{-2^{-M}}\right)$
else
$FT_{INT} = B$
$V = NIP\left(B^{-2^{-M}}\right)$
$FT_{FRA} = NIP\left(V^{-1}\right)$
end do
$B_I = FSM\left(FT_{INT}, A_I\right)$
$B_F = FSM\left(FT_{INT}, A_F\right)$
return $\boldsymbol{B_I \times B_F}$

Supposed that K is the iteration times of NIP, it can be observed that the computing complexity of improved algorithm is straightly lower than original one. The analysis of computing complexity compared with original algorithm and the improved algorithm (α = 1) is listed in TABLE VII.

TABLE VII. THE COMPUTATION TIMES OF EACH ITERATIONS USING NEWTON'S METHOD

Equation	Additions	Multiplication
Original	$2K+K^2$	$K+2MK+2K^2$
Improved	$3K$	$3K+2MK$

V. EXPERIMENTAL RESULTS

In this paper, the testing embedded system GEC2400 used for experimental estimations is shown in Fig. 2. GEC2400 comes with S3C2440A RISC microprocessor and ARM920T core. The arithmetic instruction sets of S3C2440A RISC microprocessor do not have floating types operations and integer type divisions [8]. Hence GEC2400 embedded system is very appropriate to evaluate the performances of the original algorithm, the improved algorithm, and the baseline algorithm which the floating operations is simulated by the integer operations

The experimental coefficients used in this paper are shown in TABLE VIII. This paper only calculates the first term of $\pi^{2^{-10}}$ and $\pi^{-2^{-10}}$ in fraction part with different positive and negative exponential term whose range is from –10 to 10. The computation results as well as residuals compared with the original algorithm, the proposed algorithm, and the baseline floating-point power function are shown in Fig. 3 and Fig. 4, respectively. These figures show the computing results of the improved algorithm are much closed to the baseline.

TABLE VIII. THE EXPERIMENTAL COEFFICIENTS

Experimental coefficients	Values
Exponential term	-10 to 10
Base term	π
The bits number of fraction part (*M*)	20
Output data format	64-bit signed integer

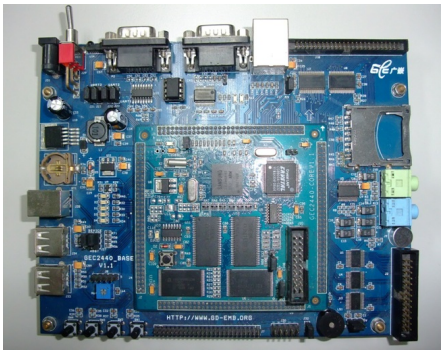

Fig. 2. The Developing Broad for the Experiments

For testing the execution time, these three algorithms are executed 2000 times with the same exponential terms value and then token their average execution time. The execution times are summarized in TABLE IX. The experimental results show that the proposed algorithm can reduce execution time about 1.8 times than the baseline one. Additionally, the performance of the proposed algorithm can reach five times higher than that of the baseline.

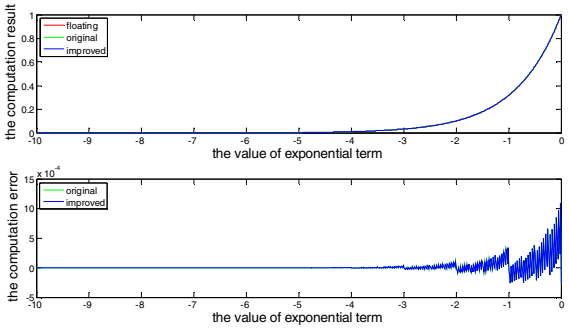

Fig. 3. The computation result of proposed algorithm compared with floating-point ones (the value of exponential term from -10 – 0)

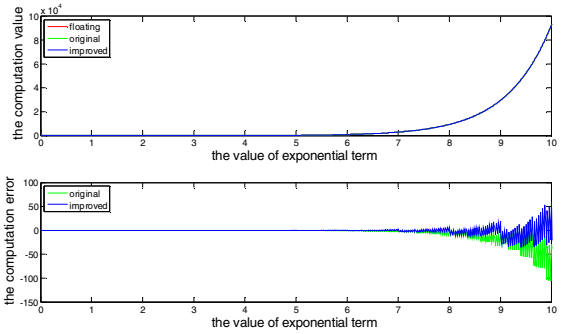

Fig. 4. The computation result of proposed algorithm compared with floating-point ones (the value of exponential term from 0 – 10)

TABLE IX. THE EXECUTION TIMES COMPARED WITH FLOATING POWER FUNCTION AND PROPOSED FIXED POINT POWER FUNCTION

Exponential term	Baseline (us)	Original algorithm		Proposed algorithm	
		Execution time (us)	Speed up ratio	Execution time (us)	Speed up ratio
≥0	2217.512	430.3898	5.1523	429.8734	5.1585
<0	2200.587	818.35	2.689	428.4667	5.1359
Average	2209.788	624.5134	3.5384	429.3134	5.1472

VI. CONCLUSIONS

A fast and efficient algorithm for fixed-point PEF using Newton's method is presented in this paper. The proposed algorithm is designed to reduce the computation time when the exponential term is positive. It makes use of FSM to reduce multiplicative computation that usually described in the fixed-point PEF and the NIP. The experimental results show that the proposed improved algorithm reduces the computation times about 5 times and also with the similar precision results compared with the baseline. The proposed fixed-point algorithm for the fixed-point PEF is very appropriate for some application on embedded system without floating mathematic instructions or for some DSP applications and IC design.

VII. REFERENCES

[1] J.-A. Piñeiro, S. F. Oberman, J.-M. Muller, and J. D.Brugera, "High-Speed Function Approximation Using a Minimax Quadratic Interpolator," *IEEE Trans. Computers*, vol. 54, pp. 304−318, Mar. 2005.

[2] C.-P. Jeannerod, H. Knochel, C. Monat, and G. Revy, "Computing Folating-Point Square Roots via Bivariate Polynomial Evaluation," *IEEE Trans. Computers*, vol. 60, pp. 214−226, Feb. 2011.

[3] I. Sajid, M. M. Ahmed, and S. G. Ziavras, "Pipelined Implementation of fixed point Square Root in FPGA Using Modified Non-Restoring Algorithm," *Proc. in ICCAE*, Sigapore, 2010, Feb. 26, pp. 226−230.

[4] C.-H. Chou, P.-C. Lin, and J.-F. Wang, "Fixed-Point acceleration of square root and logrithm using quadratic regression for HTK Kernel Modules," *Proc. in ICGEC*, Shenzhen, China, 2010, Dec. 13-15, pp. 602 −605.

[5] S. Aslan, E. Oruklu, and J. Saniie, "Realization of Area Efficient QR Factorization Using Unified Division, Square Root, and Inverse Square Root Hardware," *Proc. in EIT*, Ontario, Canada, 2009, Jun. 7-9, pp. 245 −250.

[6] N. Takagi and K. Takagi, "A VLSI Algorithm for Integer Square-Rooting," *Proc. in ISPACS*, Yonago, Japan, 2006, Dec 12-15, pp. 626−629.

[7] (2012, Aug). Wikipedia: Exponentiation by squaring [Online]. Available: Http://en.wikipedia.org/wiki/Exponentiation_by_squaring

[8] "ARM920T Technical Reference Manual," ARM Corp., 2001. Available at http://infocenter.arm.com/help/topic/com.arm.doc.ddi0151c/ARM920T_TRM1_S.pdf