

Short Paper

A Fast Base-K Logarithm with Redundant Representation

CHE WUN CHIOU AND TED C. YANG*

*Department of Electronic Engineering
Ching Yun University
Chungli, 320 Taiwan*

E-mail: cwchiou@cyu.edu.tw

**Department of Information Engineering*

*Chaoyang University of Technology
Taichung County, 413 Taiwan*

E-mail: president@mail.cyut.edu.tw

A new generation algorithm for the base-K logarithm is presented. Intermediate results for computation of the logarithm are presented in the redundant signed digit (RSD) representation, and additions and subtractions were performed without carry propagation. Based on the properties of RSD arithmetic, higher performance is obtained as compared with the conventional logarithm in the classical binary form.

Keywords: logarithm, carry-free addition, redundant representation, computer arithmetic, multiplication

1. INTRODUCTION

Logarithms have been widely used in arithmetic operations, such as multiplication and division, in many problems of digital control and digital filtering, and in almost all digital feedback applications. Over the years, several methods have been proposed for computing the logarithm of numbers to base 2. An iterative method for logarithm evaluation was presented by Majithia and Levan [1]. Later, Lo and Chen [2] proposed a generalized iterative method for the computation of logarithm from base-2 to any base-K. Chiou *et al.* [3] proposed an improvement based on a high radix that is faster than Lo and Chen's method based on a radix of 2. An important issue for logarithm evaluation by iterations is speed. When the logarithm computation is executed, the arithmetic operations involved in each iteration are multiplication, magnitude comparison, and division. Such arithmetic operations are very time-consuming due to carry-propagation. This paper describes a new iterative method for logarithm evaluation that uses a redundant binary representation to speed up multiplication, magnitude comparison, and division. For a $(u + m)$ -bit operand, with u -bit integer part and m -bit mantissa, that is based on radix-K, a time complexity of $O(u + m)$ for each iteration can be achieved.

Received September 23, 2002; revised December 1, 2003 & January 29, 2004; accepted February 18, 2004.
Communicated by Gen-Huey Chen.

2. BACKGROUND

Suppose that our objective is to generate the value of $X = \log_K A$, where $1 \leq A < K$. A popular computation process for $\log_K A$ is briefly introduced here, followed by a discussion on proposals for improvement. Let $0.x_0x_1 \dots x_n$ be the binary representation of X , where $x_i = 0$ or 1 for $0 \leq i \leq n$. Let $\log_K A = X = 0.x_0x_1 \dots x_n$. Then, by definition,

$$A = K^X = K^{0.x_0x_1 \dots x_n}.$$

By squaring both sides of the equation, we obtain

$$A^2 = K^{x_0x_1 \dots x_n}.$$

By comparing A^2 and K , the value of x_0 can be determined as follows:

- (1) if $A^2 \geq K$, then $x_0 = 1$, and set $A = A^2/K$ for the next iteration cycle; or
- (2) if $A^2 < K$, then $x_0 = 0$, and set $A = A^2$ for the next iteration cycle.

The above process is repeated iteratively. One bit of the result is generated for each such iteration. The flowchart of the algorithm is shown in Fig. 1. The arithmetic operations required for each iteration include:

- (1) multiplication, that is, $A_{j+1} = A_j * A_j$;
- (2) magnitude comparison, that is, $A_{j+1} \geq K$; and
- (3) division, that is, A_{j+1}/K if $A_{j+1} \geq K$.

Suppose that the popular shift-and-add algorithm and the add/subtract-and-shift algorithm are used in multiplication and division computations, respectively [4]. The computation time of internal addition/subtraction is linearly proportional to the word length of the operands. On average, the multiplication algorithm causes $(u + m)(2u + m)/2$ full-adder delays for both $(u + m)$ -bit operands with an m -bit mantissa. The division algorithm causes $3(u + m)(2u + m)/2$ full-adder delays on average for $(2u + m)$ -bit dividend and $(u + m)$ -bit divisor, both with an m -bit mantissa. Thus, the conventional iterative process for the logarithm computation, as shown in Fig. 1, causes $(2u + m)(5u + 5m + 4)/4$ full-adder delays on average for each iteration.

The processes of addition, subtraction, and magnitude comparison with carry propagation operate very slowly in the case of long operands. To keep the addition/subtraction and/or magnitude comparison time as short as possible, it is mandatory to overcome the propagation problem. Parallel multiplication algorithms, such as the Wallace tree [5], and fast division algorithms, such as the nonrestoring division arrays [6], have been proposed in the case of high-speed computation. However, they become rather complicated for longer operands. In [7], Nagendra *et al.* pointed out that the fastest conventional carry lookahead adder which has the best power-delay product has $O(\log_2 n)$ time complexity. An alternative solution is to use the Redundant Signed Digit (RSD)

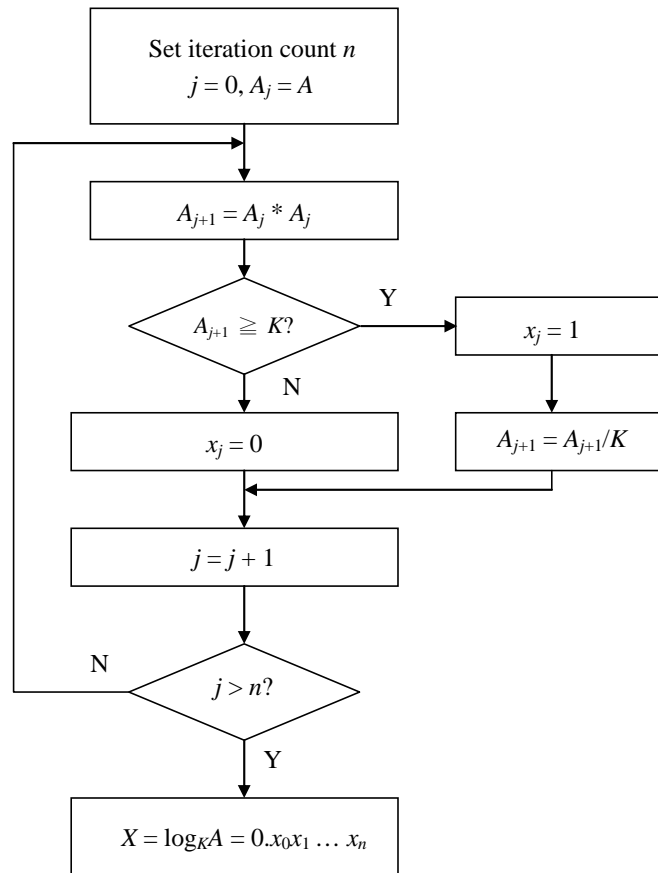


Fig. 1. Flowchart of a conventional iterative process for $\log_K A$ computation.

representation [8] instead of the ordinary binary representation. The RSD representation limits carry propagation to one position to the left during addition and subtraction operations. Therefore, two redundant binary numbers can be performed in a constant amount of time independent of the word lengths of the operands.

We consider here a redundant representation for a residue class $Z_K = \{x \mid 0 \leq x < K\}$, where $2^{u-1} \leq K < 2^u$. A $(u + m + 1)$ -bit redundant binary number with an m -bit mantissa $A = [a_u a_{u-1} \dots a_0 a_{-1} a_{-2} \dots a_{-m}]$ ($a_i = 1, 0$, or $\hat{1}$ (that is, a minus 1), $u \geq i \geq -m$) has the value $a_u * 2^u + a_{u-1} * 2^{u-1} + \dots + a_0 * 2^0 + a_{-1} * 2^{-1} + \dots + a_{-m} * 2^{-m}$. There can be several redundant binary numbers representing the same value. For example, $[0011.00]$, $[010\hat{1}.00]$, and $[1\hat{1}0\hat{1}.00]$ all represent "3". However, "0" is uniquely represented by $[00\dots 0]$. It should be pointed out that the ordinary binary representation is one of its redundant representations [9]. Therefore, no efforts are needed for conversion from the ordinary binary representation to a redundant one. However, we need a binary subtraction operation with carry propagation for reverse conversion. Let $Y = Y^* - Y^{**}$ and $Z = Z^* - Z^{**}$ be numbers in the redundant form, and let $S = S^* - S^{**}$ be the result of the op-

eration $Y + Z$ or $Y - Z$, with $y_i^*, y_i^{**}, z_i^*, z_i^{**}, s_i^*$, and $s_i^{**} \in \{0, 1\}$. Fig. 2 shows two types of generalized full adders with weighted inputs for RSD arithmetic. The RSD addition scheme for $S = Y + Z$ is shown in Fig. 3, and the subtraction scheme for $S = Y - Z$ is shown in Fig. 4. Subtraction can be performed by permuting the numbers Z^* and Z^{**} .

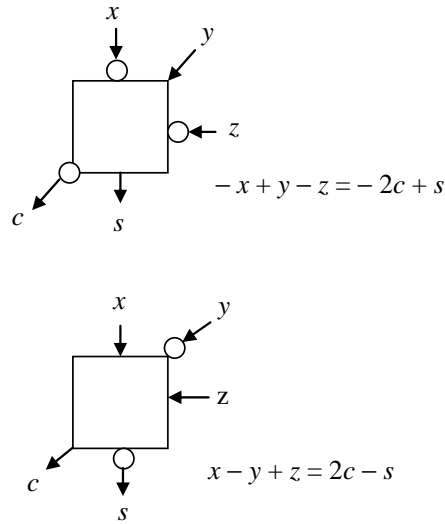


Fig. 2. Full adders for RSD addition/subtraction.

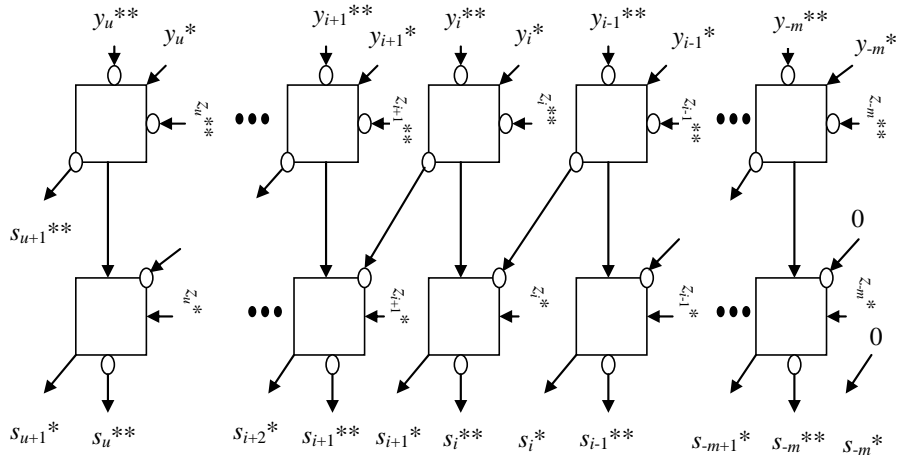


Fig. 3. RSD addition for $S = Y + Z$.

In this study, the shift-and-add multiplication and the add/subtraction-and-shift division are used for the computation of $\log_k A$. As mentioned above, the convenient addition and subtraction of numbers in the classical binary form is usually the bottleneck for speed improvement due to carry propagation. In this paper, we present a new base-K

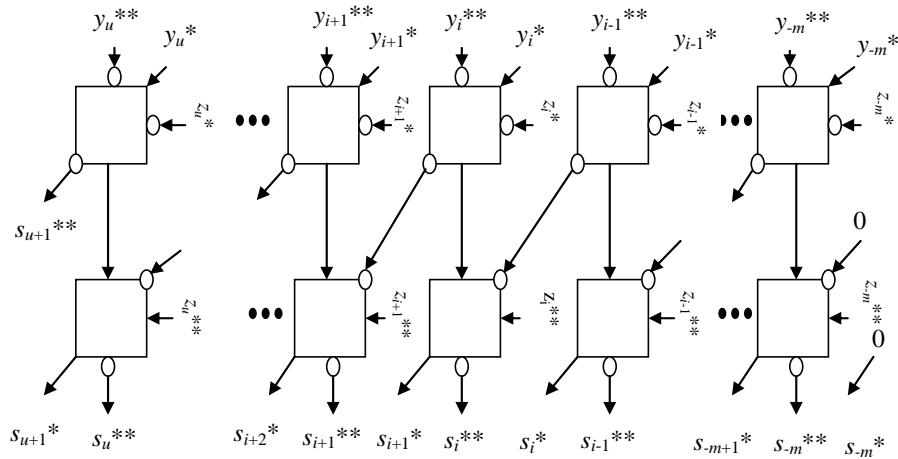


Fig. 4. RSD subtraction for $S = Y - Z$.

logarithm based on the RSD representation to solve this problem. Owing to the incorporated redundancy and the carry-propagation-free addition and subtraction, multiple additions of two redundant binary numbers can be performed in parallel in a constant amount of time independent of the word lengths of the operands. Furthermore, a time-saving “division-during-multiplication” algorithm, in which each subtraction step for division is embedded in the repeated multiply-addition step, is suggested.

3. THE PROPOSED METHOD

First of all, the addition and subtraction operations with the RSD representation will be briefly described. The proposed algorithm for base-K logarithm will then be presented, followed by a description of its operational procedure. An analysis on efficiency is also included in this section.

The adding of two redundant binary numbers (for example, $S = Y + Z$) involves two steps. For a given digit position i , the first step inputs three operand digits, y_i^{**} , y_i^* , and z_i^{**} , and generates an intermediate sum p_i and carry c_{i+1} , satisfying the equation $-y_i^{**} + y_i^* - z_i^{**} = -2c_{i+1} + p_i$. The second step adds z_i^* and the intermediate sum p_i produced by the first step and carry c_i from the next-lower-order position, and outputs s_{i+1}^* and s_i^{**} , satisfying the equation $p_i - c_i + z_i^* = 2s_{i+1}^* - s_i^{**}$. This is shown in Fig. 3. In the case of subtraction, the three operands, y_i^{**} , y_i^* , and z_i^* , are inputs in the first step, and the two outputs p_i and c_{i+1} are generated by the following equation: $-y_i^{**} + y_i^* - z_i^* = -2c_{i+1} + p_i$. In the second step, the inputs are z_i^{**} , p_i , and c_i , and the outputs are s_{i+1}^* and s_i^{**} based on the following equation: $p_i - c_i + z_i^{**} = 2s_{i+1}^* - s_i^{**}$. The subtraction scheme is shown in Fig. 4. The addition and subtraction operations are fully parallel, thus requiring no carry propagation. In other words, adding and subtracting two redundant binary numbers takes a constant amount of time, independent of the data width.

Now, let us consider the repeated multiplication-addition method. The augend Y and the addend Z are $(u + m + 1)$ -bit redundant binary numbers with an m -bit mantissa and

satisfy $-K < Y < K$ and $-K < Z < K$, respectively. We calculate the sum S , which is also an $(u + m + 1)$ -bit redundant binary number and satisfies $-K < S < K$ and $S = Y + Z \bmod K$. The following two stages are used to perform this modular addition. In the first stage, $S = Y + Z$ is carried out and $-2K < S < 2K$. The second stage is a correction stage used to keep S in the range of $-K$ and K . First, the three MSBs (most significant bits) of S are tested to identify the range, using the value of q , in which S resides. The decision table based on the three MSBs is shown in Table 1. Then, we add either K , 0 , or $-K$ to S , according to whether q is $\hat{1}$, 0 , or 1 , respectively.

Table 1. Decision table for three MSBs.

inputs			output
s_{u+1}	s_u	s_{u-1}	q
1	0	1	1
1	0	0	1
1	0	$\hat{1}$	1
1	$\hat{1}$	1	1
1	$\hat{1}$	0	1
0	1	1	1
0	1	0	1
1	$\hat{1}$	$\hat{1}$	0
0	1	$\hat{1}$	0
0	0	1	0
0	0	0	0
0	0	$\hat{1}$	0
0	$\hat{1}$	1	0
0	$\hat{1}$	$\hat{1}$	0
$\hat{1}$	1	1	0
0	$\hat{1}$	0	$\hat{1}$
$\hat{1}$	1	0	$\hat{1}$
$\hat{1}$	1	$\hat{1}$	$\hat{1}$
$\hat{1}$	0	1	$\hat{1}$
$\hat{1}$	0	0	$\hat{1}$
$\hat{1}$	0	$\hat{1}$	$\hat{1}$

- (1) If $q = \hat{1}$, this means that the following condition exists: $S < 0$. In other words, S is in the range: $-2K < S < 0$. We add K to S to satisfy $-K < (S = S + K) < K$. The corresponding bit in the quotient is $\hat{1}$.
- (2) If $q = 0$, S satisfies $-K < S < K$ and no correction step is needed. The corresponding bit in the quotient is 0 .
- (3) If $q = 1$, this means that the following condition exists: $S > 0$. In other words, S is in the range: $0 < S < 2K$. We subtract K from S to satisfy $-K < (S = S - K) < K$. The corresponding bit in the quotient is 1 .

The base-K logarithm by use of RSD arithmetic is shown in the following algorithm:

Algorithm-RSD (The base-K logarithm with RSD arithmetic)

/* Compute $\log_K A = X$, $1 \leq A < K$ */

/* Let $0.x_0x_1x_2 \dots x_n$ be the binary representation of X */

/* Let $a_{u-1} \dots a_1a_0.a_{-1}a_{-2} \dots a_{-m}$ be the binary representation of A */

/* Let A_j be the value for A in the j 'th iteration cycle */

/* Let $a_{ju}a_{j(u-1)} \dots a_{j1}a_{j0}.a_{j(-1)}a_{j(-2)} \dots a_{j(-m)}$ be the RSD representation of A_j */

Step 1: Initialize $j = 0$, $A_j = A$.

Step 2: Initialize $P = 0$, $Q = 0$, $i = u$.

Step 3: Double P and Q , i.e., $P = 2P$, $Q = 2Q$.

Step 4: If P is in the range $-K < P < K$, then no actions are performed and go to step 5.

If P is in the following range $P > 0$, and both $P = P - K$ and $Q = Q + 1$ are compute using RSD arithmetic.

If P is in the range $P < 0$, then both $P = P + K$ and $Q = Q - 1$ are compute using RSD arithmetic.

Step 5: If $a_{ji} = 0$, then go to step 6.

If $a_{ji} = 1$, then $P = P + A_j$ using RSD arithmetic.

If $a_{ji} = \hat{1}$, then $P = P - A_j$ using RSD arithmetic.

If P is in the range $-K < P < K$, then no actions are performed.

If P is in the range $P > 0$, then both $P = P - K$ and $Q = Q + 1$ are calculated using RSD arithmetic.

If P is in the range $P < 0$, then both $P = P + K$ and $Q = Q - 1$ are calculated using RSD arithmetic.

Step 6: Decrement i by 1; if $i \geq 0$, then go to step 3.

Step 7: Set $P1 = A_j$ and repeat step 8 to step 9 m times.

Step 8: Set $P1$ with the half of $P1$.

Step 9: If $a_{ji} = 1$, then $P = P + P1$ using RSD arithmetic.

If $a_{ji} = \hat{1}$ then $P = P - P1$ using RSD arithmetic.

Step 10: If P is in the range $-K < P < K$, the no actions are performed.

If P is in the range $P > 0$, then both $P = P - K$ and $Q = Q + 1$ are calculated using RSD arithmetic.

If P is in the range $P < 0$, then both $P = P + K$ and $Q = Q - 1$ are calculated using RSD arithmetic.

Step 11: Convert Q using RSD representation into the ordinary binary representation.

If $Q < 0$, then compute $A_{j+1} = P + K$ using RSD arithmetic.

If $(Q = 0)$, then compute $A_{j+1} = P$.

If $Q \geq 1$, then compute $x_j = 1$ else $x_j = 0$.

If $x_j = 0$, then go to step 14.

If $x_j = 1$, then set $q = 1$ and repeat step 12 m times.

Step 12: Double P and set q with half of q .

If P is in the range $-K < P < K$, then no actions are performed.

If P is in the range $P > 0$, then both $P = P - K$ and $Q = Q + q$ are calculated using RSD arithmetic.

If P is in the range $P < 0$, then both $P = P + K$ and $Q = Q - q$ are calculated using RSD arithmetic.

Step 13: $A_{j+1} = Q$.

Step 14: Increment j by 1; if $j \leq n$, then go to step 2; otherwise stop.

A $(u + m + 1)$ -bit RSD adder/subtractor is employed in our algorithm. One RSD addition/subtraction causes 2 full-adder delays. The proposed algorithm has about $O(n * (u + m))$ time complexity, while the conventional one has $O(n * (u + m) * \log_2(u + m))$ time complexity.

4. CONCLUSIONS

A new algorithm for base- K logarithm computation has been described. In this algorithm, numbers are represented in a redundant representation, and additions and subtractions are carried out without carry propagation. The intermediate results are represented in the redundant form, and the operands are presented in the ordinary binary representation. This feature can reduce the number of additions/subtractions required in the execution of division-during-multiplication. Magnitude comparison, which is required in the conventional approach with the ordinary form, is no longer needed in our algorithm. In comparison with the existing iterative logarithm with the ordinary binary form, the new algorithm has a time complexity of $O(n * (u + m))$, where n and $(u + m)$ are the bit lengths of X and K , respectively. Although the hardware complexity of our algorithm is twice of that of the conventional one, our algorithm is suitable for both software and hardware implementation.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable comments.

REFERENCES

1. J. C. Majithia and D. Levan, "A note on base-2 logarithm computations," in *Proceedings of IEEE*, Vol. 61, 1973, pp. 1519-1520.
2. H. Y. Lo and J. L. Chen, "A hardwired generalized algorithm for generating the logarithm base- K by iteration," *IEEE Transactions on Computers*, Vol. C-36, 1987, pp. 1363-1367.
3. C. W. Chiou, J. M. Lin, and T. C. Yang, "On the fast generation of base- K logarithm," *International Journal of Computer Mathematics*, Vol. 30, 1989, pp. 133-141.
4. K. Hwang, *Computer Arithmetic/Principles, Architecture, and Design*, New York, Wiley, 1979.

5. C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, Vol. EC-13, 1964, pp. 14-17.
6. J. C. Majithia, "Nonrestoring binary division using a cellular array," *Electronics Letters*, Vol. 6, 1970, pp. 303-304.
7. C. Nagendra, M. J. Irwin, and R. M. Owens, "Area-time-power tradeoffs in parallel adders," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 43, 1996, pp. 689-702.
8. A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, Vol. EC-10, 1961, pp. 389-400.
9. N. Takagi, H. Yasurra, and S. Yajima, "High-speed VLSI multiplication algorithm with a redundant binary addition tree," *IEEE Transactions on Computers*, Vol. C-34, 1985, pp. 789-796.

Che Wun Chiou (邱綺文) received his B.S. degree in Electronic Engineering from Chung Yuan Christian University in 1982, the M.S. degree and the Ph.D. degree in Electrical Engineering from National Cheng Kung University in 1984 and 1989, respectively. From 1990 to 2000, he was with the Chung Shan Institute of Science and Technology in Taiwan. He joined the Department of Electronic Engineering, Ching Yun University in 2000. He is currently as Dean of Department of Continuing Education in Ching Yun University. His current research interests include fault-tolerant computing, computer arithmetic, parallel processing, and cryptography.

Ted Chun-Chung Yang (楊濟中) received his B.S. degree in Electrical Engineering from National Cheng Kung University in Taiwan and Ph.D. degree in Computer Science from Illinois Institute of Technology in Chicago, U.S.A. He had engaged in research and development at Lockheed Research Laboratory, Bell Laboratories, and Motorola. He joined the faculty of Feng Chia University (FCU) in 1982 and was appointed FCU President during 1988-1995. He was General Director of the Computer and Communications Research Laboratories/ITRI in Hsinchu, Taiwan, from 1997 to 2000. Currently on leaving from FCU, he is the president of the Chaoyang University of Technology in Taiwan. His research interests include computer architecture, fault-tolerant systems, logic and system simulations, and communication systems.