Error analysis of a powering method and a novel square root algorithm

Sherif A. Tawfik, Hossam A. H. Fahmy
Electronics and Communications Department, Cairo University,
Cairo, Egypt.

Abstract - This paper presents a complete error analysis for a novel square root hardware implementation. The analysis includes the powering method used for the initial approximation and the higher order Newton-Raphson square root iterations. Both theoretical and algorithmic error analysis are presented and compared. The algorithmic analysis provides a more accurate error estimate which reduces the size of the memory required in the initial approximation stage to less than half its original size.

Keywords— **High-Speed arithmetic, Square root, Division, Iterative methods, Error analysis, Powering method.**

I. Introduction

Any general purpose system complying to the IEEE floating point standard [iee85] must provide an implementation of the division and square root operations. Both are also important operations for signal processing. Although they occur less frequently than the floating point addition, subtraction or multiplication, improving their performance improves the system performance for many applications [OBE 97].

The algorithms for the division and square root calculations are grouped in two broad categories: subtractive and multiplicative. In the subtractive methods, the convergence rate is linear and the main operation is a subtraction coupled with a multiplication by a single digit. On the other hand, in the multiplicative methods, the convergence rate is quadratic and the main operation is a full multiplication. Multiplicative algorithms are also called functional iteration algorithms because they start with an initial approximation of the result and iterate on a function involving additions and multiplications to get the final solution. The iterations use, in general, the Newton-Raphson root-finding algorithm either from the first or a higher order. The convergence rate of the multiplicative algorithm is even better than quadratic if a higher order function is used.

This work presents the error analysis for the higher order square root multiplicative algorithm [TAW 05] and for the initial approximation method, the powering method, that is employed in the algorithm. Both theoretical and algorithmic error analysis are presented and compared. The theoretical method is based on Taylor's expansion. The algorithmic method on the other hand is based partially on an exhaustive search. The algorithmic method gives a tighter error bound. In fact it gives the actual maximum error of the algorithm. This more accurate error analysis reduces the area and delay of the resulting circuits.

The remainder of the paper is organized as follows: Section II presents the theoretical error analysis of the powering method while section III presents the algorithmic error analysis. Section IV reviews the square root algorithm presented in [TAW 05] while its error analysis is presented in section V. The case of the odd exponent is treated in section VI. Finally, section VII concludes the paper.

II. The powering method and its theoretical error analysis

The powering method introduced by Takagi [TAK 97] can be used to generate the power of an operand, i.e., X^a for an operand X and a given, fixed a. The power a has the form $\pm 2^b$ where b is any integer or $\pm 2^{b_1} \pm 2^{b_2}$ where b_1 is any integer and b_2 is any non-negative integer. We focus in this section on the special case $a = -2^{-b}$ in which b is a non-negative integer. The same analysis can be applied to the other cases as well.

If the significand of the number has the binary representation $X = [1.x_1x_2x_3\cdots x_n]$ such that $X \in [1, 2[$ and we define the two quantities $p = [1.x_1x_2x_3\cdots x_m]$ and $q = 2^{-m-1}[x_{m+1}.x_{m+2}x_{m+3}\cdots x_n]$ then $X = p + q$. Simple mathematical manipulations yield:

$$\begin{aligned}
X^a &= (p+q)^a && (1)\\
&= (p + 2^{-m-1} + q - 2^{-m-1})^a && (2)\\
&= (p+2^{-m-1})^a \left(1 + \frac{q-2^{-m-1}}{p+2^{-m-1}}\right)^a && (3)\\
&\approx (p+2^{-m-1})^a \left(1 + a\frac{q-2^{-m-1}}{p+2^{-m-1}}\right) && (4)\\
&\approx (p+2^{-m-1})^{a-1}(p+2^{-m-1}+a(q-2^{-m-1})) && (5)\\
&\approx (p+2^{-m-1})^{-(2^{-b})-1}\ (p+2^{-m-1} \\
&\qquad -2^{-b}(q-2^{-m-1})) && (6)
\end{aligned}$$

If the bits of p, excluding the leading '1', index a table to get the coefficient $c = (p + 2^{-m-1})^{-(2^{-b})-1}$ then $X^a \approx c\tilde{X}$ where $\tilde{X} = (p + 2^{-m-1} - 2^{-b}(q - 2^{-m-1}))$. Expanding $\tilde{X}$ gives

$$\begin{array}{llllllll}
\tilde{X} = & 1.x_1 & \cdots x_m 1 & 0 \cdots & 0 & 1 & 0 & \cdots 0 \\
 & -0.0 & \cdots 0 \;\; 0 & 0 \cdots & 0 & x_{m+1} & x_{m+2} & \cdots x_n \\
\tilde{X} = & 1.x_1 & \cdots x_m 1 & 0 \cdots & 0 & 1 & 0 & \cdots 1 \\
 & +1.1 & \cdots 1 \;\; 1 & 1 \cdots & 1 & \overline{x}_{m+1} & \overline{x}_{m+2} & \cdots \overline{x}_n \\
\tilde{X} = & 1.x_1 & \multicolumn{6}{l}{\cdots x_m \overline{x}_{m+1} x_{m+1} \cdots x_{m+1} x_{m+1} \overline{x}_{m+2} \cdots \overline{x}_n} \\
 & +2^{-n-b} & & & & & &
\end{array}$$

There are b bits of the value x_{m+1} between $\overline{x}_{m+1}$ and $\overline{x}_{m+2}$. If the 2^{-n-b} term is neglected then a simple rewiring and bitwise inversion in X yields $\tilde{X}$.
Equation 4 retains only the first two terms of the infinite series. This infinite series is alternating (each term has opposite sign to the previous one) and the terms are decreasing in magnitude. Therefore, the approximation error due to the series expansion, ϵ_s, has the first truncated term multiplied by the external factor $(p + 2^{-m-1})^a$ as an upper bound.

$$\epsilon_s < \frac{(-2^{-b})(-2^{-b}-1)}{2} \frac{(q - 2^{-m-1})^2}{(p + 2^{-m-1})^{2^{-b}+2}}$$

$$\epsilon_s < \frac{(2^{-b})(2^{-b}+1)}{2} \frac{(2^{-m} - 2^{-m-1})^2}{(1 + 2^{-m-1})^{2^{-b}+2}}$$

$$\epsilon_s < \frac{(2^{-b})(2^{-b}+1)}{2} \times 2^{-2m-2} \qquad (7)$$

Another error arises from storing a finite precision value in the table to approximate the coefficient c. The most significant bit of c is always 0 and it is not stored. Hence, if the table width is t bits, the truncation error is less than 2^{-t}. The part of the approximation error due to the truncation of c is equal to the truncation error multiplied by the maximum value of $\tilde{X}$. Since $\tilde{X} < 2$, the total approximation error is

$$\epsilon < \frac{(2^{-b})(2^{-b}+1)}{2} \times 2^{-2m-2} + 2^{-t+1} \qquad (8)$$

The size of the lookup table giving this error is $2^m \times t$ bits. Equation 8 gives the theoretical error bound of the algorithm.

III. The algorithmic error analysis of the powering method

The error can be expressed as

$$\epsilon = (p+q)^{-2^{-b}} - c(p + 2^{-m-1} - 2^{-b}(q - 2^{-m-1})) \quad (9)$$

where $c = (p + 2^{-m-1})^{-2^{-b}-1}$ truncated to t bits. To find the error bound, we might search exhaustively for the values of p and q that produce the maximum error. However, an exhaustive search is not possible because of the huge number of combinations and the length of time required to carry this search. The elimination of q from equation 9 makes the exhaustive search feasible since the number of potential values for p is small. To eliminate q from equation 9 we must find the value of q that gives the maximum error analytically. Fortunately, ϵ is a differentiable function with respect to q. Hence, we use the first and second derivatives in order to determine the value of q at which ϵ attains its maximum value.

$$\frac{\partial \epsilon}{\partial q} = (-2^{-b})(p+q)^{-2^{-b}-1} + c(2^{-b}) \qquad (10)$$

$$\frac{\partial^2 \epsilon}{\partial q^2} = (2^{-b})(2^{-b}+1)(p+q)^{-2^{-b}-2} \qquad (11)$$

Equation 11 indicates that $\frac{\partial^2 \epsilon}{\partial q^2} > 0$. Therefore, the extremum point obtained from equation 10 is a local minimum. The value of q giving the maximum error is thus one of the end points of the interval i.e. $q = 0$ or $q = 2^{-m} - 2^{-n}$. Substituting these two values in equation 9, we get two error functions ϵ_1 and ϵ_2 that are functions of p only.

$$\epsilon_1 = p^{-2^{-b}} - c(p + 2^{-m-1} + 2^{b-m-1}) \qquad (12)$$

$$\begin{aligned}\epsilon_2 = {} & (p + 2^{-m} - 2^{-n})^{-2^{-b}} \\ & - c(p + 2^{-m-1} + 2^{b-n} - 2^{b-m-1})\end{aligned} \qquad (13)$$

We next search for the value of p that maximizes each of these two functions and select the greatest of the two errors as the maximum approximation error of the algorithm.
Table I lists both the theoretical and algorithmic error bounds for the square root reciprocal ($b = 1$) approximation. From the table we can see that the algorithmic error analysis gives a tighter bound. In fact, it gives the actual maximum error i.e. the best bound.

IV. Square root algorithm

In this section we review our proposed [TAW 05] square root implementation. The algorithm is from the multiplicative category and it computes the square root for the double precision format. The algorithm employs the powering method for the initial approximation of the square root reciprocal. It then performs one iteration of the second order Newton-Raphson algorithm followed by a multiplication by the operand

TABLE I
COMPARISON BETWEEN THE THEORETICAL AND ALGORITHMIC ERROR BOUNDS FOR THE SQUARE ROOT RECIPROCAL OPERATION

m	t	Theoretical	Algorithmic
6	17	$2^{-14.67}$	$2^{-15.06}$
7	18	$2^{-16.2}$	$2^{-16.85}$
8	21	$2^{-18.67}$	2^{-19}
9	23	$2^{-20.67}$	2^{-21}
10	24	$2^{-22.2}$	$2^{-22.7}$

to get an approximation to the square root. Finally the algorithm rounds this result according to one of the four IEEE Rounding modes. The initial approximation parameters m and t have the values 8 and 21 respectively.

Fig. 1 presents the proposed architecture. The box labeled 'g' is a temporary register to hold the intermediate results of the calculation. The main block in the figure is the fused multiply add (FMA) unit. Given $X = p + q$, the steps of the algorithm are:

1) From p and the lookup table $\rightarrow c$,
from $X \rightarrow \tilde{X}$,
$c\tilde{X} \rightarrow y_0$

2) $y_0 \times y_0 \rightarrow r$

3) $1 - r \times X \rightarrow d$

4) $\frac{3}{8}d + \frac{1}{2} \rightarrow u$

5) $1 + d \times u \rightarrow v$

6) $y_0 \times v \rightarrow y_1$

7) $y_1 \times X \rightarrow z$. For RZ or RM modes, round z to the nearest value using the guard bits and jump to step 8. For RP mode round z to the neasrest value using the guard bits and jump to step 9. For RN mode truncate the guard bits of z and jump to step 10.

8) If $z^2 - X > 0$, then result $= z - 1ulp$. Otherwise, result $= z$.

9) If $z^2 - X < 0$, then result $= z + 1ulp$. Otherwise, result $= z$.

10) If $(z + 0.5ulp)^2 - X > 0$, then result $= z$. Otherwise, result $= z + 1ulp$.

In steps 8, 9 and 10 the FMA acts as a multiply subtract unit by adding the two's complement of the third operand.

V. Error analysis of the square root algorithm

The error in the approximation is due to two sources: the first is the approximation method while the second source is the truncation of the intermediate results in the eight steps of the algorithm. We present an error analysis for each source in the following subsections.

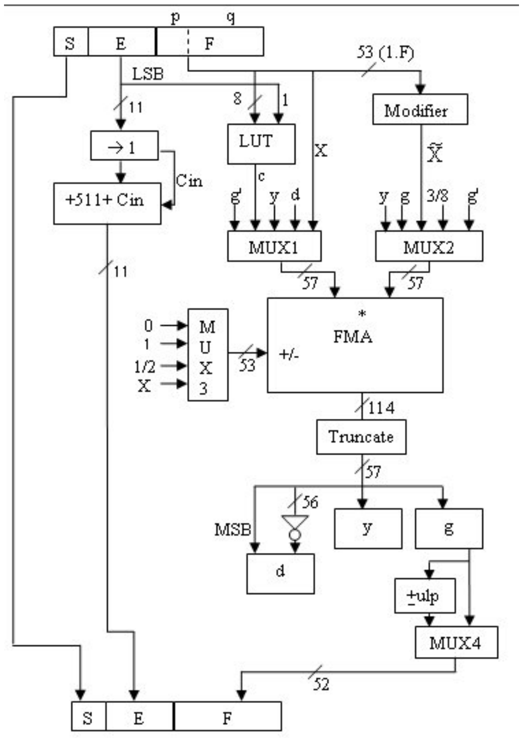

Fig. 1. Block diagram of the proposed architecture.

A. Error analysis of the approximation method

If the initial approximation is called y_0 and we define the quantity $d = 1 - y_0^2 X$ then

$$\frac{1}{\sqrt{X}} = \frac{y_0}{\sqrt{1-d}} \tag{14}$$

$$\frac{1}{\sqrt{X}} \approx y_0 \left(1 + \frac{1}{2}d + \frac{3}{8}d^2 + \cdots\right) \tag{15}$$

Keeping k terms gives a k^{th} convergent algorithm which yields a number of correct bits in iteration $i+1$ that is k-times the number of correct bits in iteration i. Specifically, if only three terms are kept— which is our case— such that $y_1 = y_0 \left(1 + \frac{1}{2}d + \frac{3}{8}d^2\right)$ and since $y_0 = X^{-0.5} - \epsilon$ then

$$d = 2\epsilon\sqrt{X} - \epsilon^2 X \tag{16}$$

$$d^2 = 4\epsilon^2 X - 4\epsilon^3 X\sqrt{X} + \epsilon^4 X^2 \tag{17}$$

$$y_1 = \frac{1}{\sqrt{X}} - \frac{5}{2}\epsilon^3 X + \frac{15}{8}\epsilon^4 X\sqrt{X} - \frac{3}{8}\epsilon^5 X^2 \tag{18}$$

The error after one iteration is dominated by the negative term $-\frac{5}{2}\epsilon^3 X$. The algorithm thus gives almost three times the number of correct bits each iteration and guarantees that $y_1 < \frac{1}{\sqrt{X}}$. We multiply the final approximation by the operand X in order to obtain an approximation to the square root from the square root reciprocal. Therefore, the final error is given by

$$\epsilon_f = \frac{5}{2}\epsilon^3 X^2 \quad (19)$$

$$\epsilon_f = \frac{5}{2}(max(\epsilon_1, \epsilon_2))^3 (p+q)^2 \quad (20)$$

$$\epsilon_f < \frac{5}{2}(max(\epsilon_1, \epsilon_2))^3 (p+2^{-m}-2^{-n})^2 \quad (21)$$

where the results of our algorithmic analysis are used. Equation 21 gives the final approximation error as a function of p only. Hence we can search for the maximum error by using an exhaustive search. The maximum final error for the case of $m = 8$ and $t = 21$ has the value $\epsilon_f = 2^{-55.7}$. If we use the theoretical error analysis bound the final error takes the value $\epsilon_f = 2^{-52.7}$ This means that the theoretical bound underestimates the accuracy of the algorithm by 3 bits.

B. *Truncation Error analysis of the square root algorithm*

To guard against the accumulation of the truncation error we add L guard bits to the FMA inputs. We choose the value of L such that the sum of the approximation and truncation errors lies in a suitable interval for the given rounding mode.
We define the truncation error as the difference between the untruncated value and the truncated value. Each operand of the FMA has $53 + L$ bits. One bit lies to the left of the binary point and $52+L$ bits lie to the right of the binary point. The product of two such operands has a total of $106 + 2L$ bits. Two bits lie to the left of the binary point and the rest to the right. We truncate this product to $53 + L$ bits by throwing away the most significant bit which is always zero and the least $52 + L$ bits. The error that results from this truncation has a maximum value of 2^{-52-L}.
We calculate the truncation error in the first seven steps of the algorithm. The truncation error of step 1 is included in the error analysis of the initial approximation.

Step	Error term	Error interval	
2 :	ϵt_r	$[0, 2^{-52-L}]$	(22)
3 :	ϵt_d	$-\left(max(X) \times \epsilon t_r + [0, 2^{-52-L}]\right)$ $= [-3 \times 2^{-52-L}, 0]$	(23)
4 :	ϵt_u	$\frac{3}{8} \times \epsilon t_d + [0, 2^{-52-L}]$ $= [-\frac{9}{8}2^{-52-L}, 2^{-52-L}]$	(24)
5 :	ϵt_v	$max(d) \times \epsilon t_u + max(u) \times \epsilon t_d + [0, 2^{-52-L}]$ $= [-\frac{3}{2}2^{-52-L}, 2^{-52-L}]$	(25)
6 :	ϵt_{y1}	$max(y0) \times \epsilon t_v + [0, 2^{-52-L}]$ $= [-\frac{3}{2}2^{-52-L}, 2 \times 2^{-52-L}]$	(26)
7 :	ϵt_z	$max(X) \times \epsilon t_{y1} + [0, 2^{-52-L}]$ $= [-3 \times 2^{-52-L}, 5 \times 2^{-52-L}]$	(27)

As indicated in step 7 of our algorithm, for the case of round to zero, minus infinity or plus infinity we round the guard bits. The error from this rounding lies in the interval $[-2^{-53}, (0.5 - 2^{-L}) \times 2^{-52}]$. The sum of the total truncation error and the approximation error is $[-3 \times 2^{-52-L} - 2^{-53}, (0.5 + 2^{2-L}) \times 2^{-52} + 2^{-55.7}]$. Therefore, the minimum number of guard bits that guarantees correct rounding is four.
When $L = 4$, the total error is $[-0.6875 \times 2^{-52}, 0.827 \times 2^{-52}]$. For the case of RZ or RM rounding modes we divide the total error interval into the two intervals: $[-0.6875$ ulp$, 0[$ and $[0, 0.827$ ulp$]$. If the total error lies in the first interval then the correct rounded result is $z - 1$ ulp and if it lies in the second interval then the correct rounded result is z. As for the RP rounding mode we divide the total error interval into the two intervals: $[-0.6875$ ulp$, 0]$ and $]0, 0.827$ ulp$]$. If the total error lies in the first interval then the correct rounded result is z and if it lies in the second interval then the correct rounded result is $z + 1$ ulp. We determine in which of the two intervals the total error lies by comparing the argument X with z^2.
For the case of round to nearest, we truncate the guard bits. The truncation error is $[0, (1 - 2^{-L}) \times 2^{-52}]$. The sum of the total truncation error and the approximation error is $[-3 \times 2^{-52-L}, (1 + 2^{2-L}) \times 2^{-52} + 2^{-55.7}]$. Therefore, the minimum number of guard bits that guarantees correct RN rounding is four. When $L = 4$, the total error is $[-0.1875$ ulp$, 1.33$ ulp$]$. The computed machine number z is less than the true result by no more than 1.33 ulp and greater than the true result by no more than 0.19 ulp. Since we are computing the square root, it is impossible for the true result to be exactly equal to $z + 0.5$ulp. Otherwise, the original argument would be $(z+0.5\text{ulp})^2$ which requires more bits to represent than is available for the argument. Hence,

the round to nearest even and the round to nearest up modes are equivalent in this case. If the total error lies in the interval $[-0.1875 \text{ ulp}, 0.5 \text{ ulp}[$ the round to nearest result is z. If, on the other hand, the total error lies in the interval $]0.5 \text{ ulp}, 1.33 \text{ ulp}[$ the round to nearest result is $z + 1$ ulp. The comparison of the argument X with $(z + 0.5 \text{ ulp})^2$ determines which of the above two intervals containes the result. We then get the correct rounded result which is either z or $z+1$ ulp.

VI. The case of the odd exponent

The case of the odd unbiased exponent needs further elaboration. In this case, we subtract one from the exponent and multiply the mantissa by two in order to leave the argument unaltered and at the same time make the exponent even. We call the new mantissa X_{odd} ($X_{odd} \in [2, 4[$) and the initial approximation coefficient c_{odd}.
Changes in the initial approximation stage:

$$\begin{aligned} c_{odd} &= c_{even} \times (2^{-\frac{3}{2}}) \\ \tilde{X}_{odd} &= 2 \times \tilde{X}_{even} \end{aligned}$$

Since we are only interested in the product of c_{odd} and $\tilde{X}_{odd}$, we can multiply c_{odd} by 2 and divide $\tilde{X}_{odd}$ by 2 leaving their product unchanged. Hence,

$$c_{odd} = c_{even} \times \frac{1}{\sqrt{2}} \tag{28}$$

$$\tilde{X}_{odd} = \tilde{X}_{even} \tag{29}$$

Changes in steps 2 and 6:
In both steps we shift the product one bit to the left (multiplication by two) before storing it. Hence in the immediately following steps, step 3 and 7 respectively, we replace X_{odd} by the original X. This shifting takes place at the output of the multiplier using a multiplexer.
This way the inputs to the multiplier are the same for both cases of odd or even exponent. The only change is in the value of the initial approximation coefficient and in the way we store the output of the FMA in the intended register.
The error analysis must be reperformed. Following the same error analysis method we get:

- the approximation error $= [0, 0.223 \times 2^{-52}]$,
- the total truncation error is the same and is equal to $[-0.1875 \text{ ulp}, 1.25 \text{ ulp}]$ for RN rounding mode and $[-0.6875 \text{ ulp}, 0.75 \text{ ulp}]$ for the remaining rounding modes,
- the total error is equal to $[-0.1875 \text{ ulp}, 1.47 \text{ ulp}]$ for RN rounding mode and $[-0.6875 \text{ ulp}, 0.97 \text{ ulp}]$ for the remaining rounding modes.

The reliance on the theoretical error analysis solely leads to the following results:

- the initial approximation error $\epsilon = 2^{-18.95}$
- the approximation error $= [0, 2^{-51.52}]$,
- the total truncation error is the same and is equal to $[-0.1875 \text{ ulp}, 1.25 \text{ ulp}]$ for RN rounding mode and $[-0.6875 \text{ ulp}, 0.75 \text{ ulp}]$ for the remaining rounding modes,
- the total error is equal to $[-0.1875 \text{ ulp}, 2.64 \text{ ulp}]$ for the RN rounding mode and to $[-0.6875 \text{ ulp}, 2.14 \text{ ulp}]$ for the remaining rounding modes.

With only the theoretical analysis, we would increase the initial approximation accuracy by employing a bigger table ($m = 9$ and $t = 22$)in order to ensure correct rounded results in all rounding modes. Therefore the more accurate algorithmic error analysis reduces the memory requirements of the algorithm to less than half of what the theoretical analysis predicts. This decrease in the memory requirement not only reduces the area of the circuit but also its time delay and power consumption.

VII. Conclusions

We have presented a review to the powering method and high order Newton-Raphson square root algorithms. A detailed error analysis is given. A new algorithmic error analysis is developed and presented. This new approach reveals that the algorithmic analysis is more accurate than the theoretical estimate. A direct consequence of this more accurate analysis is a substantial decrease in the memory requirement of the initial approximation stage. Had we used the theoretical error analysis in the square root algorithm we would have needed over twice the initial approximation memory that we use.

REFERENCES

[iee85] *IEEE Standard for Binary Floating-Point Arithmetic, (ANSI/IEEE Std 754-1985)*, IEEE press, New York, NY, august 1985.

[OBE 97] Oberman S. F., Flynn M. J., *Design Issues in Division and Other Floating Point Operations*, *IEEE Transactions on Computers*, vol. 46, n2, p. 154–161, febrary 1997.

[TAK 97] Takagi N., Generating a Power of an Operand by a Table Look-up and a Multiplication, *Proceedings of the 13th IEEE Symposium on Computer Arithmetic, Asilomar, California, USA*, p. 126–131, july 1997.

[TAW 05] Tawfik S. A., Fahmy H. A. H., *Square Root and Division: An Improved Algorithm and Implementation*, *To be published*, 2005.