Verification of Hardware combining Multiplication, Division and Square Root

Colin D. Walter

Computation Department, U.M.I.S.T.,
PO Box 88, Sackville Street, Manchester M60 1QD, U.K.
e-mail: cdw@sna.co.umist.ac.uk

Index Terms: *Computer arithmetic, multiplication, division, square root, redundant number systems, recurrence relations.*

Abstract

This note provides a proof, corrections and minor generalisation of algorithms used by M. Ercegovac and T. Lang for implementing a hardware module that combines multiplication, division and square root. A consequence of the proof is the supply of initialisation conditions and bounds for register contents. Some subtleties required of an implementation are also noted.

1 Introduction.

In [3] Ercegovac and Lang describe a hardware design which combines algorithms for multiplication, division and square root and generates output digit serially. The work described here emanates from problems one of my project students had in proving the correctness of the design: without seeing a step-by-step derivation of the algorithm it was difficult to know how to set about verifying it.

The same authors in [2] explain how "on-line" algorithms might be derived. These are algorithms in which the inputs are consumed and the outputs generated digit serially at the same rate, most significant bit first. The techniques in [2] are applicable in general for iterative arithmetic, including here, where most inputs are provided entirely at initialisation, and the outputs produced one digit at a time. However, at least for the square root, further ideas are necessary to complete the details of the proof.

Recent advances in technology have triggered much research into arithmetic by enabling increased speed to be gained at the cost of relatively cheap hardware. The square root algorithm has, of course, been extensively studied. In particular, a version of the algorithm here appears in [7], and ones using higher bases appear in [4] and [8], where different initialisation complexities arise. The use of Newton-Raphson iteration for the square root is studied in [6]. Although it has quadratic rather than linear convergence so that it is apparently faster, the method involves more arithmetic per iteration (a full length squaring) which destroys any advantage.

On the back of the square root case, at little further expense we have included proofs for the multiplication and division algorithms. This makes sense because some initialisation detail is missing in [3] and indeed the former produces an answer which is out by a factor of 2. Combined modules for multiplication, division and square root have also been studied in [10] where the radix is 4. The multiplication method used there produces output digit serially starting as usual with the least significant digit, but here it is in the opposite order, in order to match the output of the other two algorithms. This type of multiplication has already been discussed in [5], whilst division follows the normal paper and pencil method.

We start with a derivation of some recurrence relations. This proves only the partial correctness of the algorithms. Total correctness requires also a demonstration of convergence; more specifically, that run-time errors are not caused by overflow. This second part of the proof reveals and exhibits some subtle detail of what is required from a hardware implementation, including bounds on register contents, and enables us to note some potential problems if the original description is varied in any way.

2 Notation.

Since reals are generally expressed in sign/mantissa/exponent form, we will assume all mantissa representations are shifted to make the first non-zero digit correspond to a negative power of the radix 2, normally to 2^{-1}. As calculations with the exponents and signs are relatively straight-forward here, only operations on the mantissas are considered further.

We adopt notation in which uppercase characters are used to denote real numbers and lowercase characters to denote digits. So we consider number representations $M = \sum_{t=1}^{\infty} m_t 2^{-t}$, and sequences such as $M[j] = \sum_{t=1}^{j} m_t 2^{-t}$ $(j \geq 0)$ where m_t is a digit. The latter arise for inputs and outputs whose digits are supplied or generated serially.

The addition of two numbers can lead to arbitrarily long carry propagation if a standard non-redundant binary output is demanded. However, considerable speedup is derived by allowing a redundant representation for the output since carry propagation can then be limited, thus enabling digit operations to be performed in parallel. The algorithms here use a representation for the outputs of an addition with digits in the range 0..2.

Speed is also achieved by ignoring less significant digits. This leads to less precision in the digits of the outputs of the algorithms. The inexactitude is recovered in subsequent digits by allowing a wider range of digits for the outputs, namely $-1..1$. By incorporating an appropriate constant delay into such outputs, it is possible to convert to a different redundant representation, or generate two converging non-redundant representations which bracket the value (see [1]).

3 The Recurrence Relations.

The derivation of the recurrence relations of Ercegovac and Lang [3] is straightforward. The key step is to define a *residual error* or *partial remainder* $W[j]$ for the j^{th} iteration. This is a measure of the difference between the desired operation on the inputs so far and the output generated so far. It is defined iteratively, and at each iteration another output digit is chosen to approximately minimise its next value. Convergence of the output to the right answer is guaranteed if this scaled residual error is bounded, and this in turn relies on appropriate normalisation of the inputs and definition of the output digits.

Following the notation of [3] we denote inputs by X and Y and outputs by P (product), Q (quotient) and S (square root). The basic definitions are most conveniently written:

$$0 = X\times Y - P \qquad 0 = X - Y\times Q \qquad 0 = X/2 - (S\times S)/2$$

By taking approximations to about j places, they yield the expressions

$$\begin{aligned} 2^{-j}W[j] &= X\times Y[j{+}1] - P[j] \\ 2^{-j}W[j] &= X - Y\times Q[j] \\ 2^{-j}W[j] &= X/2 - S[j]\times S[j]/2 \end{aligned}$$

for $j \geq 0$, which define $W[j]$ as the weighted *residual error* at the end of the j^{th} iteration. Since the output digits have only positive indices, the initial output values satisfy

$$P[0] = Q[0] = S[0] = 0$$

In [3] it is assumed the inputs are in normalised non-redundant form, so that $Y[1] = y_1 2^{-1} = 1/2$ in the equation for multiplication. Hence the initial values of W are respectively:

$$\begin{aligned} W[0] &= X/2 \\ W[0] &= X \\ W[0] &= X/2 \end{aligned}$$

The recurrence relations are obtained by using the formula $M[j] = M[j{-}1] + 2^{-j}m_j$ applied to Y, P, Q and S in the relations for $W[j]$ and then subtracting the expressions for $W[j{-}1]$. Thus, for $j \geq 1$,

$$\begin{aligned} W[j] &= 2W[j{-}1] + y_{j+1}X/2 - p_j \\ W[j] &= 2W[j{-}1] - q_jY \\ W[j] &= 2W[j{-}1] - s_jS[j{-}1] - 2^{-j-1}{s_j}^2 \end{aligned}$$

The quantities on the right side of these relations show what data must be available as input to the iteration, and hence most memory requirements in an implementation. Hardware for implementing the algorithms needs two registers for redundant representations (W and one of X, Y or S) plus a cyclically rotatable binary register in which to store 2^{-j}.

Comparing these algorithms with those described in [3], the recurrence relations for division and square root agree, but not that for multiplication: every appearance of X in the multiplication algorithm here appears as $X/2$ in [3], so that $P = Y{\times}X/2$ is calculated there rather than the true product – even the bound of $X < 1/2$ here appears as the bound $X < 1$ there. So we must record this as an error in [3], although the normalisation of the output there should automatically correct the unwanted factor of 2. The initialisation for $W[0]$, which is omitted from [3], is given above.

4 Output Digits and Residual Error Bounds.

The output digits are chosen to minimise the next residual error as far as possible using a minimal amount of computation. The redundant representation allows some flexibility in their choice so that an estimate $W[j]'$ of $W[j]$ is only needed to two places after the point. As W is the output from a carry save adder it has digits in the range 0..2. Consequently,

$$W[j]' \quad \leq \quad W[j] \quad < \quad W[j]' + 1/2 \tag{1}$$

Thus $W[j]' + 1/4$ is, on average, the best approximation to $W[j]$. So Ercegovac and Lang define:

$$\begin{array}{rcl} p_j & = & \left\{ \begin{array}{r} 1 \text{ if } W[j{-}1]' \geq \quad 0 \\ 0 \text{ if } W[j{-}1]' < -1/4 \text{ or } \; -1/2 \\ -1 \text{ if } W[j{-}1]' \leq -3/4 \end{array} \right\} \\ q_j & = & sign(\ W[j{-}1]' + 1/4\ ) \\ s_j & = & sign(\ W[j{-}1]' + 1/4\ ) \end{array}$$

where $sign$ maps onto the digit set $\{-1, 0, +1\}$ in the obvious way. By choosing a more accurate approximation W' it is possible to increase the number of zero output digits so that the iterative step reduces from an addition to a simple left shift, and performance can then be improved (see [9]).

Establishing the convergence of the algorithms requires choosing exactly the right bounds for the residual error. They are obtained by selecting expressions which will only just work in the proofs.

Theorem 1. If $0 \leq X < 1/2$, $1/2 \leq Y < 1$ and Y has the usual non-redundant binary representation then the residual error in the multiplication algorithm is bounded by

$$-1 \quad \leq \quad W[j] \quad < \quad 1 - X/2 \quad \leq \quad 1 \qquad \text{for all} \quad j \geq 0 \ .$$

Theorem 2. If $0 \le X < 1/2$ and $1/2 \le Y < 1$ then the absolute value of the residual error in the division algorithm is bounded by

$$|W[j]| \quad \le \quad Y \quad < \quad 1 \qquad \text{for all} \quad j \ge 0 \ .$$

Theorem 3. (cf (7) in [7].) If $1/4 \le X < 1$ and either $W[0]$ is in non-redundant binary form when s_1 is computed or $s_1 = 1$ is forced, then

i) $W[j] - 2^{-j-1} \ < \ S[j] \ < \ 1$ for all $j \ge 0$;

ii) $-1 \ < \ -S[j] \ \le \ W[j] - 2^{-j-1}$ for all $j \ge 1$.

For the first two theorems the case of $j = 0$ follows directly from the initial conditions. The rest follows easily by induction on j using (1) in a case by case analysis of output digit values. For the square root algorithm the same line of reasoning works providing that a sufficiently good lower bound on $S[j]$ is known. This is given by the lemma below, but first note that the extra initialisation property is needed. Thus, if $X = .01...$ produces the representation $W[0] = \bar{1}.1122...$ $(= X/2)$ at the time s_1 is calculated, then $s_1 = 0$ would result. Since such X and S would then satisfy $X \ge 1/4$ and $S < 1/2$, the relation $X = S^2$ could not hold. However, if $W[0]$ has standard binary form then $s_1 = 1$ results. So for a correct algorithm $s_1 = 1$ must always hold independently of the representation of $W[0]$ and it may need to be forcefully initialised.

We need to distinguish two cases: we call s_j an *initial zero* of S if $s_i = 0$ whenever i satisfies $1 < i \le j$ and a *non-initial zero* of S if $s_j = 0$ but $s_i \ne 0$ for some i satisfying $1 < i < j$.

Lemma. Given the initialisation of Theorem 3,

i) If s_j is an initial zero then $S[j] = S[1] = 1/2$;

ii) If s_j is the first non-zero digit with $j > 1$ then $s_j = 1$;

iii) If s_j is a non-initial zero then $S[j] > 1/2 + 2^{-j-1}$.

Proof. Part (i) is immediate from $s_1 = 1$. Now suppose $s_i = 0$ for $1 < i < j$. Then $W[j-1] = 2^{j-2}(X - 2^{-2}) \ge 0$. Hence $W[j-1]' > W[j-1] - 1/2 \ge -1/2$. So $s_j \ne -1$, giving part (ii). However, if s_j is a non-initial zero and s_{i-1} is the last initial zero then

$$S[j] \ \ge \ S[i-1] + 2^{-i} - (2^{-i-1} + ... + 2^{-j+1}) \ = \ 1/2 + 2^{-j+1} \ > \ 1/2 + 2^{-j-1}.$$

Corollary. With the initialisation given in the theorems and earlier, all three algorithms converge to the correct answer, and $|W[j]| \le 1$ for all $j \ge 0$.

This is clear, given $S[j] \le 1 - 2^{-j}$. It is worth remarking that $|W[j]| = 1$ could actually arise, but only for multiplication (for example, take $X = 1/4$ and $Y = 1/2$ with $j \ge 3$).

5 Summary and Conclusions.

We have proved the total correctness of the algorithms used in [3] subject to a correcting factor of 2 in the case of multiplication, and a guarantee that $s_1 = 1$ in the case of the square root. The recurrence relations for them were derived in §3, where the initial conditions, omitted from [3], are clarified. The definition of the output digits is given at the start of §4. The inputs are subject to the range and representation restrictions stated in the theorems. As these are less demanding than those in [3] the theorems have been slightly generalised. The serially produced output is always within 1 in the most recent digit position of being the correct infinite precision answer. Finally, in all cases, the residual error $W[j]$ has absolute value at most 1 and so hardware registers of the right size can now be supplied.

References

[1] Milos D. Ercegovac & Tomas Lang, "On-the-fly conversion of redundant into conventional representations", *IEEE Trans. Comput.*, vol. **C-36**, pp. 895-897, July 1987.

[2] Milos D. Ercegovac & Tomas Lang, "On-Line Arithmetic: A Design Methodology and Applications in Digital Signal Processing", *VLSI Signal Processing III*, R. W. Brodersen, H. S. Moscovitz eds., pp. 252-263, IEEE Press, New York, 1988.

[3] Milos D. Ercegovac & Tomas Lang, "Implementation of a Module combining Multiplication, Division and Square Root", *Proc. IEEE Intern. Symp. on Circuits and Systems*, 1989, pp. 150-153.

[4] Milos D. Ercegovac & Tomas Lang, "Radix-4 Square Root Without Initial PLA", *IEEE Trans. Comput.*, vol. **C-39**, pp. 1016-1024, August 1990.

[5] Milos D. Ercegovac & Tomas Lang, "Fast Multiplication Without Carry-Propagate Addition", *IEEE Trans. Comput.*, vol. **C-39**, pp. 1385-1390, November, 1990.

[6] Reza Hashemian, "Square Root Algorithms for Integer and Floating point Numbers", *IEEE Trans. Comput.*, vol. **C-39**, pp. 1025-1029, August 1990.

[7] Stanislaw Majerski, "Square Rooting Algorithms for High Speed Digital Circuits", *IEEE Trans. Comput.*, vol. **C-34**, pp. 724-733, August 1985.

[8] Paolo Montuschi & Luigi Ciminiera, "On the efficient implementation of higher radix square root algorithms", *Proc. 9th IEEE Symposium on Computer Arithmetic*, Santa Monica, CA, pp. 154-161, 1989.

[9] Paolo Montuschi & Luigi Ciminiera, "Reducing Iteration Time When Result Digit is Zero for Radix 2 SRT Division and Square Root with Redundant Remainders", *IEEE Trans. Comput.*, vol. **42**, pp. 239-246, February 1993.

[10] J. H. Zurawski & J. B. Gosling, "Design of a high-speed square root, multiply and divide unit", *IEEE Trans. Comput.*, vol. **C-36**, pp. 13-23, 1987.