On-the-fly Rounding for Division and Square Root

Miloš D. Ercegovac and Tomas Lang

Computer Science Department
School of Engineering and Applied Science
University of California, Los Angeles

Abstract

In division and square root implementations based on digit-recurrence algorithms the result is obtained in digit-serial form, from most significant digit to least significant. Moreover, to reduce the complexity of the result-digit selection and to allow the use of redundant addition, the result-digit has values from a signed-digit set. As a consequence, the result has to be converted to conventional representation. This conversion can be done on-the-fly as the digits are produced, without the use of a carry-propagate adder. In this paper we describe how to modify this conversion process so that the result is rounded. The resulting operation is faster than what is done conventionally because no carry-propagate addition is needed. We describe three rounding methods; they differ in the rounding error and the hardware and time required.

1. Introduction

In division and square root implementations based on digit-recurrence algorithms the result is obtained in digit-serial form, from most significant digit to least significant. Moreover, to reduce the complexity of the result-digit selection and to allow the use of redundant addition, the result-digit has values from a signed-digit set [HWAN78]. As a consequence, the result has to be converted to conventional representation. We have shown in [ERCE87] that this conversion can be done on-the-fly as the digits are produced, without the use of a carry-propagate adder, which is needed in the traditional approach. In this paper we describe how to modify this conversion process so that the result is rounded. Without loss of generality we discuss a sequential implementation - the scheme applies equally well to linear array (combinational) implementations.

We assume that operands and result are represented in sign and magnitude; consequently, we operate only with the magnitudes and the signs are processed separately. Of course, the method can be adapted to other representations, such as true-and-complement.

Traditionally, rounding to nearest is done as described for example in [FAND87]. It requires the following steps:

1) An additional digit of the result is obtained for the rounding.

2) Restoration step. Since the last partial remainder can be negative, a restoration step is required to produce a positive remainder. To achieve this, the sign of the last partial remainder is determined. Since the adder used in the recurrence is redundant (for example carry-save), the sign has to be obtained from this redundant representation; the process is similar in delay, but simpler in amount of hardware, to a carry-propagate addition that converts the partial remainder to conventional representation. If the sign is negative, the result is decremented by means of a carry-propagate addition.

3) Rounding step. Finally, the (unrounded) result is rounded to nearest by possibly incrementing it by 1. This incrementation requires a carry-propagate addition.

The overhead of this process is high, both in hardware and time. We describe three rounding methods that reduce the overhead; they differ in the rounding error and the hardware and time required. In summary, the methods have the following characteristics:

1) A method that produces the correct rounding to nearest as specified by the IEEE standard [COON80]. This method requires the computation of the sign of the remainder. We show that the result decrement and increment operations can be incorporated in the on-the-fly conversion process, so that no carry-propagate addition is needed.

2) A method which provides unbiased rounding with a somewhat larger error than rounding to nearest. The implementation of this method does not require the sign of the remainder, nor does it require the incrementation of the result. Consequently, it is simple to implement. It is useful in applications that can accept the somewhat larger error.

3) A compromise between the previous two methods uses an estimate of the sign of the remainder. This estimate is computed using a few most-significant bits of the remainder, reducing the time for this computation. The resulting implementation is fast and produces an error in between that of the previous methods.

Note that the other rounding schemes of the IEEE Standard can be implemented in a similar manner. Note also that the conventional value of the remainder is not obtained; if it is required a carry-propagate adder has to be used.

2. Rounding to nearest

We discuss now how the steps mentioned before can be incorporated into the on-the-fly conversion.

1) An additional digit of the result is obtained. This requires one more iteration. However, to incorporate the restoration step, the resulting digit is not used in the same manner as the previous ones for the on-the-fly conversion. Let us call $Q[n]$ the converted result with n digits and p_{n+1} the $(n+1)$–th digit of the result.

2) The sign of the partial remainder is determined, so that

$$sign = \begin{cases} 1 & \textbf{if } \textit{remainder is negative} \\ 0 & \textbf{otherwise} \end{cases}$$

Since a negative partial remainder makes it necessary to decrement the result, the correct value of the $(n+1)$–th digit becomes $(p_{n+1} - sign)$. If the value of p_{n+1} is in the range $[-a, a]$, the correct digit is in the range $[-a-1, a]$.

Using this correct digit, a conventional representation of the $(n+1)$-digit result could be obtained by the on-the-fly conversion process. We would obtain the following digit-vectors (this will not have to be computed explicitly, as explained later):

$$Q[n+1] = \begin{cases} (Q[n], (p_{n+1} - sign)) & \text{if } (p_{n+1} - sign) \geq 0 \\ (QM[n], (r - |p_{n+1}| - sign)) & \text{if } (p_{n+1} - sign) < 0 \end{cases}$$

where $QM[n] = Q[n] - r^{-n}$ is the second form produced in the on-the-fly conversion process [ERCE87].

3) The rounding to nearest operation then produces the result $q[n]$ with n digits, as described in Table 1, where $QT[n]$ is $Q[n+1]$ truncated to n digits and $QP[n] = Q[n] + r^{-n}$.

Table 1. Rounding-to-nearest process

$(p_{n+1} - sign)$	$q[n]$
$[r/2, a]$	$QT[n] + r^{-n} = Q[n] + r^{-n} = QP[n]$
$[0, r/2)$	$QT[n] = Q[n]$
$[-r/2, 0)$	$QT[n] + r^{-n} = QM[n] + r^{-n} = Q[n]$
$[-a-1, r/2)$	$QT[n] = QM[n]$

From Table 1, the process is implemented as

$$q[n] = \begin{cases} QP[n] & \text{if } (p_{n+1} - sign) \geq r/2 \\ Q[n] & \text{if } -r/2 \leq (p_{n+1} - sign) < r/2 \\ QM[n] & \text{if } (p_{n+1} - sign) < -r/2 \end{cases} \quad (1)$$

In addition, to have unbiased rounding to nearest, the result $q[n]$ is jammed to even when $|p_{n+1}| = r/2$ and the last remainder is equal to zero.

The forms $Q[n]$ and $QM[n]$ are produced for the on-the-fly conversion [ERCE87]. To be able to do the rounding we need to produce also the form $QP[n]$.

The updating of the three forms is performed when each signed-digit of the result is produced. Let us call p_{k+1} the digit produced during the k–th iteration with values in the range $\{-a, -a+1, \ldots -1, 0, 1, \ldots a-1, a\}$. The updating of the forms $Q[k]$ and $QM[k]$ is the same as for the on-the-fly conversion [ERCE87]. That is,

For $k>1$

$$Q[k+1] = \begin{cases} (Q[k], p_{k+1}) & \text{if } p_{k+1} \geq 0 \\ (QM[k], (r - |p_{k+1}|)) & \text{if } p_{k+1} < 0 \end{cases} \quad (2a)$$

$$QM[k+1] = \begin{cases} (Q[k], (p_{k+1} - 1)) & \text{if } p_{k+1} > 0 \\ (QM[k], ((r-1) - |p_{k+1}|)) & \text{if } p_{k+1} \leq 0 \end{cases} \quad (2b)$$

For the third form $QP[k]$ we have,

$$QP[k+1] = \begin{cases} (Q[k], (p_{k+1} + 1)) & \text{if } -1 \leq p_{k+1} \leq r-2 \\ (QM[k], (r - |p_{k+1}| + 1)) & \text{if } p_{k+1} < -1 \\ (QP[k], 0) & \text{if } p_{k+1} = r-1 \end{cases} \quad (2c)$$

Note that the last condition for $QP[k+1]$ is only applicable when the digit set of the signed-digit is maximally redundant $(a = r-1)$.

Implementation

A possible implementation of the rounding-to-nearest scheme is shown in Figure 1. It consists of three left-shift registers to keep the forms $Q[k]$, $QM[k]$, and $QP[k]$ and logic to generate the digit to concatenate and the loading controls. Table 2 describes the logic for the radix-4 case, with signed-digit set $p_k = \{-3, -2, -1, 0, 1, 2, 3\}$.

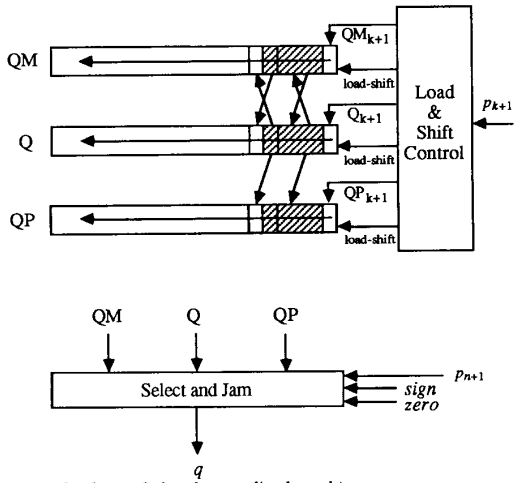

Figure 1. Scheme for conversion/rounding - to - nearest

Table 2. Updating for radix-4 case

p_{k+1}	$Q[k+1]$	$QM[k+1]$	$QP[k+1]$
0	$(Q[k],0)$	$(QM[k],3)$	$(Q[k],1)$
1	$(Q[k],1)$	$(Q[k],0)$	$(Q[k],2)$
-1	$(QM[k],3)$	$(QM[k],2)$	$(Q[k],0)$
2	$(Q[k],2)$	$(Q[k],1)$	$(Q[k],3)$
-2	$(QM[k],2)$	$(QM[k],1)$	$(QM[k],3)$
3	$(Q[k],3)$	$(Q[k],2)$	$(QP[k],0)$
-3	$(QM[k],1)$	$(QM[k],0)$	$(QM[k],2)$

The rounding is performed by the selection described by expressions (1). In addition it is necessary to have a network to detect the sign of the remainder from its redundant representation. The most straightforward implementation uses a carry-propagate adder to convert to nonredundant representation; this is specially attractive if a carry-propagate adder exists in the arithmetic unit anyhow for other purposes. However, in some cases it might be better to have a special network for this sign detection, either because a carry-propagate adder is not part of the unit or because connection to this adder is slow or complicates the bussing structure. The main component of this sign detection network is the generation of the carry into most-significant bit; this network follows the standard carry-skip or carry-lookahead techniques.

A possible carry-skip implementation of this sign-detection network is shown in Figure 2. In it the carry-save remainder is divided into blocks of 4 bits. For a block, a network computes C_{out}, the carry generated by the block, and P, which is 1 when the block propagates a carry. The worst-case delay of this scheme can be further reduced by a partitioning into variable-length blocks.

Finally, the detection of zero remainder is needed for the unbiased rounding to nearest. This detection can follow a conversion of the final remainder to irredundant, if such a conversion is used for sign detection, or another special circuit can be used [CORT88], as described now and shown in Figure 2. We define z_i such that

$$z_i = \begin{cases} 1 & \text{if } s_j = 0 \text{ for } j \geq i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where s_j is the j^{th} bit of the sum of WS and WC. That is, $z_i = 1$ if the sum is zero up to the i^{th} bit. Then,

$$z_{i-1} = (p_{i-1} \oplus k_i) \cdot z_i \tag{4}$$

where $p_i = WS_i \oplus WC_i$ and $k_i = WS_i' WC_i'$.

A zero-skip implementation generates signals N and P and Z for each block such that

$$N = \prod k_i \tag{5a}$$

$$P = \prod p_i \tag{5b}$$

$$Z = \prod (p_{i-1} \oplus k_i) \tag{5c}$$

and produces

$$z_{out} = (Nc_{in}' + Pc_{in} + Z) z_{in} \tag{6}$$

The zero-remainder condition is represented by the output $z_{out} = 1$ of the left-most block.

3. Rounding without sign detection

As a second method for rounding, we consider the case in which the sign of the remainder is not detected. This results in a simpler and faster implementation, but with a somewhat larger error.

To determine an appropriate rounding scheme let us consider the error committed with respect to the infinite precision result q when an $(n+1)$-digit result is computed. We then round to produce the minimum error. Since the result is produced in signed-digit form and the sign of the remainder is not known, we obtain $q \in [L, U]$ such that

$$q > L = Q[n] + (p_{n+1} - \frac{a}{r-1}) r^{-(n+1)} \tag{7a}$$

$$q < U = Q[n] + (p_{n+1} + \frac{a}{r-1}) r^{-(n+1)} \tag{7b}$$

These expressions are illustrated in Figure 3 for both signs of p_{n+1}. To minimize the error, we choose among $Q[n]$, $QM[n]$, and $QP[n]$ the one producing the smallest maximum error. From Figure 3, we see that the choice is as follows:

$$q[n] = \begin{cases} Q[n] & \text{if } |p_{n+1}| < r/2 \quad (\textit{Region A}) \\ QP[n] & \text{if } p_{n+1} > r/2 \quad (\textit{Region B}) \\ QN[n] & \text{if } p_{n+1} < -r/2 \quad (\textit{Region C}) \end{cases} \tag{8}$$

Notice that this expression excludes the values $|p_{n+1}| = r/2$. For these values, the error bounds for the three choices are

$q[n]$	$p_{n+1} = r/2$	$p_{n+1} = -r/2$
$Q[n]$	$(\frac{r}{2} + \frac{a}{r-1}) r^{-(n+1)}$	$-(\frac{r}{2} + \frac{a}{r-1}) r^{-(n+1)}$
$Q_+[n]$	$-(\frac{r}{2} + \frac{a}{r-1}) r^{-(n+1)}$	
$Q_-[n]$		$(\frac{r}{2} + \frac{a}{r-1}) r^{-(n+1)}$

This last table indicates that there are two choices that produce the same error, and that the error is larger than that for rounding to nearest ($2^{-1} r^{-n}$). This verifies that exact rounding to nearest is not possible without knowing the sign of the remainder. However, an unbiased rounding scheme is obtained if for these cases we balance the signs of the errors. This happens if we choose $Q[n]$ for both $p_{n+1} = r/2$ and $p_{n+1} = -r/2$. The resulting rounding scheme is

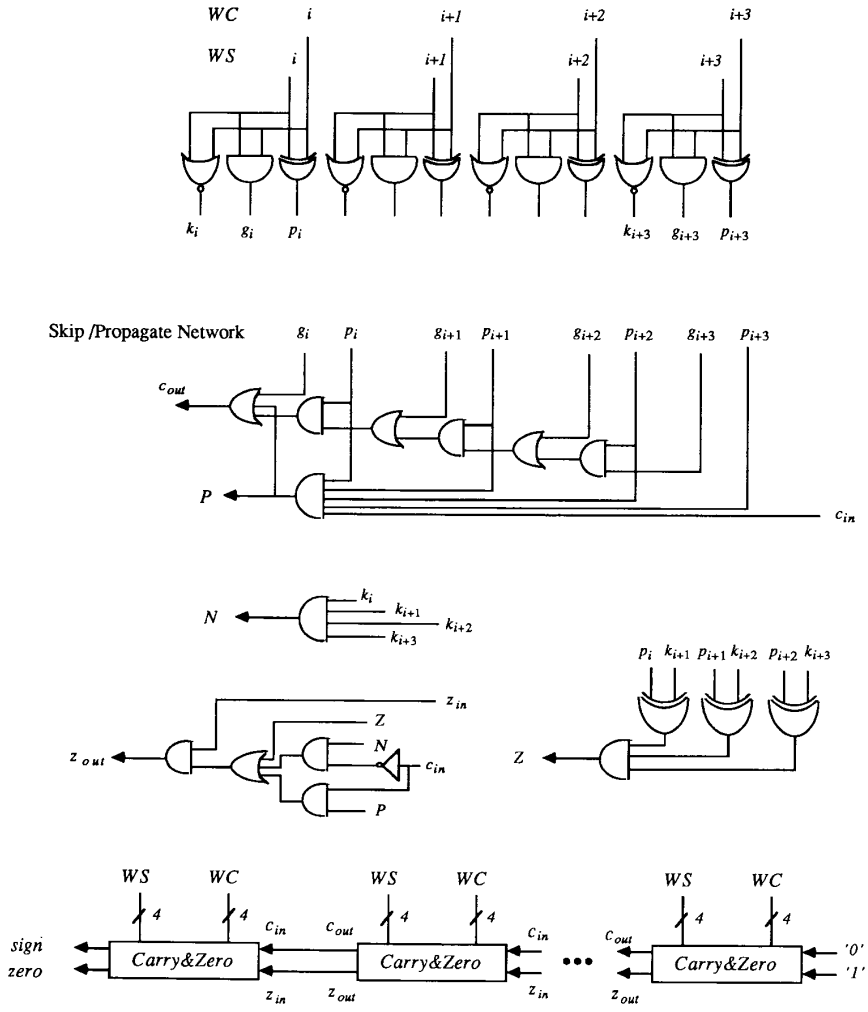

Figure 2. Sign and Zero Detection Network

$$q[n] = \begin{cases} Q[n] & \text{if } |p_{n+1}| \le r/2 \\ Q_+[n] & \text{if } p_{n+1} > r/2 \\ Q_-[n] & \text{if } p_{n+1} < -r/2 \end{cases} \quad (9)$$

Note that, in contrast to rounding-to-nearest, the scheme is unbiased without need of the detection of the case $r/2,0,0,0....$

The radix-2 case is specially simple, resulting in (unbiased) signed truncation. That is,

$$q[n] = Q[n] \quad (10)$$

The error in this case is bounded by $\pm 2^{-n}$.

Implementation

The implementation of this scheme uses the same module to compute $Q/QM/QP$ as in the first method. However, it does not need the remainder sign-detection nor the zero detection. For radix-2, there is no need of QP either.

4. Rounding with estimate of the sign

The previous method is simple to implement but, specially for the radix-2 case, might have an error that is too large. To reduce the error, it is possible to use an estimate of the sign of the remainder and then apply the rounding rules of method 1. The estimate is computed using a few most-significant bits of the remainder, resulting in a reduction in cost and time with respect to the exact detection of the sign.

Specifically, if the b most-significant bits of the remainder are used the resulting error is bounded by $r^{-n}(2^{-1}+2^{-b})$. The value of b is chosen to get both an acceptable error and a suitable delay. As an example, for radix-2 and $b=8$ the error would be $2^{-n}(2^{-1}+2^{-8})$, which differs from the error for rounding to nearest by less than 1%. Moreover, the rounding using the estimate of the sign from eight bits of the remainder could be done in one short cycle, instead of about four cycles that a complete sign detection would require.

5. Example of rounding schemes

We now show an example of the three rounding schemes. The representation is radix 4 with result-digit set {-2,-1,0,1,2}. Table 3 shows the conversion when the n-th digit of the result is produced and the rounding when the digit $n+1$ is obtained.

k	$n-1$	n
$Q[k]$	$xx23$	$xx223$
$QP[k]$	$xx30$	$xx230$
$QM[k]$	$xx22$	$xx222$
p_{k+1}	-1	-2
remainder		
sum	---	01011xxx
carry	---	00101xxx
sign	---	1
sign est (4 bits)	---	0
sign est (5 bits)	---	1

The rounding process results in

rounding to nearest	$q[n]=QM[n]=xx222$
rounding without sign	$q[n]=Q[n]=xx230$
rounding with sign est (4 bits)	$q[n]=Q[n]=xx230$
rounding with sign est (5 bits)	$q[n]=QM[n]=xx222$

Summary

The proposed approach extends the on-the-fly method of converting redundant into conventional representations in digit-recurrence algorithms to obtain rounded results without carry-propagate addition. Three rounding alternatives are described differing in the error, time, and hardware. The approach, discussed in the context of sequential algorithms, is applicable in combinational arrays of linear type.

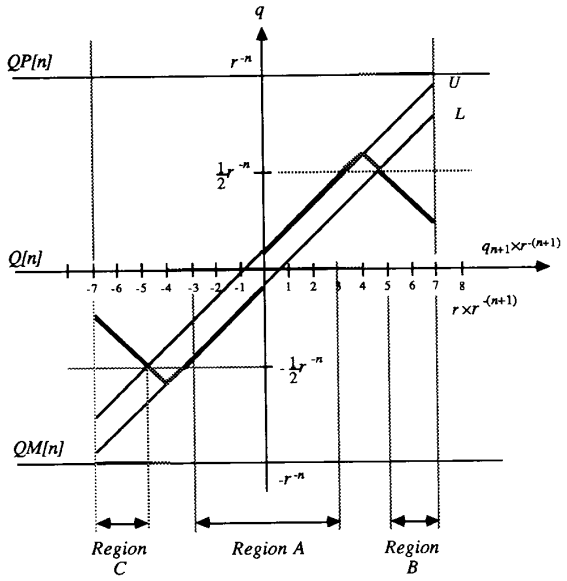

Figure 3. Rounding without Sign Detection (r = 8)

Acknowledgments This research has been supported in part by the NSF Grant No. MIP-8813340 *Composite Operations Using On-Line Arithmetic for Application-Specific Parallel Architectures: Algorithms, Design, and Experimental Studies.*

References

[COON80] J.T. Coonen, "An Implementation Guide to a Proposed Standard for Floating-Point Arithmetic", Computer, pp.68-79. 1980.

[CORT88] J. Cortadella and J.M. Llaberia "Evaluating A+B=K conditions in constant time", International Conference on Circuits and Systems, Helsinki, 1988.

[ERCE87] M.D. Ercegovac and T. Lang, "On-the-Fly Conversion of Redundant into Conventional Representations", IEEE Transactions on Computers, Vol. C-36, No.7, July 1987, pp.895-897.

[FAND87] J. Fandrianto, "Algorithm for High Speed Shared Radix-4 Division and Radix-4 Square Root," Proc. 8th Symposium on Computer Arithmetic, 1987, pp. 73-79.

[HWAN78] K. Hwang, *Computer Arithmetic*, John Wiley & Sons, 1978.