Fast Decimal Floating-Point Division

Hooman Nikmehr, Braden Phillips, *Member, IEEE*, and Cheng-Chew Lim, *Senior Member, IEEE*

***Abstract*—A new implementation for decimal floating-point (DFP) division is introduced. The algorithm is based on high-radix SRT division[1] with the recurrence in a new decimal signed-digit format. Quotient digits are selected using comparison multiples, where the magnitude of the quotient digit is calculated by comparing the truncated partial remainder with limited precision multiples of the divisor. The sign is determined concurrently by investigating the polarity of the truncated partial remainder. A timing evaluation using a logic synthesis shows a significant decrease in the division execution time in contrast with one of the fastest DFP dividers reported in the open literature.**

***Index Terms*—Binary-coded decimal (BCD), decimal floating-point (DFP) arithmetic, digit recurrence division.**

I. Introduction

PERFORMING manual calculations using decimal arithmetic is part of human nature. Typical computers, on the other hand, support binary arithmetic more readily. The ENIAC, which became operational in 1945 at the University of Pennsylvania, was one of the early attempts to use radix 10 calculations in digital computers [1]. The IBM eServer z900 seems to be the only recent processor capable of performing decimal instructions in hardware [2], [3]. However, its decimal computation capability is limited to integers operands. Recently, decimal arithmetic has become more attractive in the financial and commercial world including banking, tax calculation, currency conversion, insurance, and accounting. The following facts may explain this recent interest.

- A survey of commercial databases [4] shows that 98.6% of the numbers stored are decimal or integer while more than half of them are represented in pure decimal format.
- It is well understood that when converting between decimal and binary formats, most fractional decimal numbers are only approximately represented in binary floating-point (FP) representation and, therefore, may loose precision [5], [6]. This means that using binary FP numbers in financial applications, which cannot tolerate errors, does not necessarily guarantee correct results.
- Regulations such as that for the European Commission Directorate General II [7] specify decimal digits for currency calculations.

The importance of decimal arithmetic has led to a proposed revision to the IEEE 754 standard for FP arithmetic to include specifications for decimal arithmetic [8]. This means that although computers are still carrying out decimal FP (DFP) calculations using software libraries [9], [10] and binary FP numbers, it is likely that in the near future, many high-end processors will perform decimal operations directly on DFP operands using dedicated DFP units, which are hundreds of times faster than the software packages [11].

In this paper, a DFP division algorithm and its implementation are proposed. The algorithm is based on well-known high-radix SRT[1] division [12]–[15]. The division's partial remainder is represented in a new redundant format called decimal signed-digit (DSD) and the SRT recurrence is carried out using decimal carry-free (DCF) addition [16]. Unlike conventional dividers, which use the potential difference (PD) plot or the selection constants methods for quotient digit selection [17], the new divider is designed using comparison multiples [18].

There are three novel issues presented in this paper; employing the SRT division algorithm, which was originally developed for radices of power of 2, to implement a divider with a radix that is not presented as a power of 2, removing the PD plot (the lookup table) from the conventional implementation of the SRT algorithm and replacing it with a set of comparators followed by sign detectors, and introducing a divider that can perform decimal division faster than any available counterpart.

It should be noted that the design is optimized for speed at the expense of power and area. This is reasonable for one of the first DFP dividers in the literature; however, low-power architectures remain an open topic.

This paper is organized as follows. High-radix SRT division is discussed in Section II. Section III presents previous related works in decimal division. In Section IV, the fundamentals of DSD arithmetic are discussed. Specification of the DFP format introduced in the proposal of the IEEE 754R standard [8] is briefly explained in Section V followed by a short review of the rounding issues in DFP arithmetic. Section VI describes how high-radix SRT division can be used for implementing DFP division. Section VII gives a timing evaluation of the implementation. This paper presents a conclusion in Section VIII.

II. High-Radix SRT Division

Surveys [14] and [19] show that many VLSI implementations of FP division are based on the SRT digit recurrence division algorithms. SRT division is an iterative algorithm with linear convergence toward the quotient. In this algorithm, the quotient digit selection (QDS) function calculates a fixed number of bits of the quotient every iteration. The speed of a SRT divider is mainly determined by the complexity of the QDS function [20], which is traditionally implemented using the lookup table

Manuscript received October 6, 2005; revised January 10, 2006 and May 18, 2006.

H. Nikmehr is with the Department of Computer Engineering, Bu Ali Sina University, Hamedan, 65157-19418, Iran (e-mail: nhooman@eleceng.adelaide.edu.au).

B. Phillips and C.-C. Lim are with the School of Electrical and Electronic Engineering, University of Adelaide, Adelaide, SA 5005, Australia.

Digital Object Identifier 10.1109/TVLSI.2006.884047

[1]The SRT division algorithm is named after D. Sweeney [46], J. E. Robertson [47], and T. D. Tocher [48].

method. In this method, the QDS function is realized in a table implemented with a programable logic array (PLA) or a ROM [17].

Radix-r SRT division compliant with the IEEE 754 standard [21] is implemented using the recurrence

$$w[j+1] = rw[j] - q_{j+1}d, \qquad \text{where } j = 0, 1, \ldots, m \quad (1)$$

plus one more cycle to produce the quotient in the IEEE 754 standard. In the recurrence (1), the dividend x and the divisor d are two normalized binary numbers in the range $[0.5, 1)$ and m is the number of iterations needed to produce the bits of the precision. Also, q_{j+1} represents the quotient digit in the signed-digit (SD) redundant format [22] selected from the redundant radix-r digit set

$$\left\{\overline{a}, \overline{a-1}, \ldots, \overline{1}, 0, 1, \ldots, a-1, a\right\} \quad (2)$$

where $\overline{n} = -n$ and $\lceil r/2 \rceil \leq a \leq r-1$. In (1), the next partial remainder (PR), $w[j+1]$, is represented in a redundant format [22]. To select the correct q_{j+1}, the QDS function applies the most significant bits of $rw[j]$ and d to a lookup table. The resulting quotient digit should be such that $w[j+1]$ is always bounded as

$$-\rho d \leq w[j+1] \leq \rho d \quad (3)$$

where $\rho = (a/r-1)$ is known as the *redundancy factor*. The inequality (3) is known as the *convergence condition*. To satisfy the convergence condition in the first iteration, the first PR, $w[0]$, is initialized to $r^{-1}x$. The quotient $q = \sum_{j=1}^{m} q_j r^{-j}$ represented in SD is converted to radix-r nonredundant form using the on-the-fly conversion and rounding algorithm [23].

III. History of Decimal Dividers

Yabe *et al.* [24] present an implementation of a digit recurrence decimal division. Every iteration, limited digit numbers from the normalized divisor and the PR are applied to a prediction table to find a value for the quotient digit. If the new PR is negative, then the divisor is added to the PR (restoration step) and the incorrectly predicted quotient digit, which is represented in a redundant form, is adjusted in the next iteration. This restoring decimal division requires a full-length decimal addition per iteration. Moreover, if restoration is needed, a full-length decimal subtraction must be performed as well. Therefore, due to these two time consuming operations, the divider is very slow.

The division presented by Busaba *et al.* [3] is a simple restoring division, which like the approach of Yabe *et al.* [24], subtracts the divisor from the PR each iteration until the PR becomes negative. The algorithm restores the PR and obtains the corresponding quotient digit. This method, which is used in the IBM z900 decimal arithmetic unit, again uses a very slow full-length BCD addition/subtraction per iteration.

Yamaoka *et al.* [25] develop a nonrestoring decimal divider, which subtracts the divisor from the dividend every iteration and accumulates the number of iterations in the quotient register. It suffers from a full-length decimal addition/subtraction every iteration since the PR is represented in the conventional binary-coded decimal (BCD) format. Also, as another disadvantage, the number of iterations is unknown until a negative PR is found.

A nonheuristic decimal divider presented by Ferguson [26], determines one quotient digit every iteration. It limits the number of candidates for the quotient digit to two by normalizing both the divisor and the dividend. The normalization is performed by multiplying the dividend and the divisor by a common factor. This adds a large delay overhead to the operation. To select the correct quotient digit from the candidates, a subtraction must be performed. Using the quotient digit, the corresponding multiple of the normalized divisor is selected from a table. The multiples must be precomputed and stored in the table. This operation requires parallel full-length multiplications (or successive full-length additions/subtractions), which not only increase the decimal division execution time massively but also increase the implementation area. The divider adjusts the final quotient if the divisor is normalized at the beginning of division. This adjustment involves doubling, and several times incrementing or decrementing the quotient. The delay penalty caused by the adjustment is also rather high.

Wang and Schulte [27] claim the first DFP divider complying with the proposal of the IEEE 754R standard. The divider, which uses the Newton–Raphson iteration [14], doubles the number of quotient digits every iteration. This feature makes it very fast compared to the previously discussed decimal dividers. This improvement is achieved by using an optimized piecewise linear approximation, a modified Newton–Raphson iteration, a specialized rounding technique, and a simplified combined decimal incrementer/decrementer in the design. However, a Newton–Raphson iteration requires two multiplications every iteration. This has a negative impact on the latency of decimal division since a pipelined decimal multiplier is difficult to build and requires much more delay than decimal addition.

IV. DSD Arithmetic

Among decimal arithmetic operations, there are some complex functions such as sequential multiplication and digit recurrence division, which compute and accumulate partial results. These repetitive operations cannot be accomplished in a reasonable time without fast circuits for decimal addition. One method for implementing fast decimal adders is to take advantage of carry-free addition.

Nikmehr [18] introduces an 8-bit decimal signed-digit arithmetic for representing redundant decimal digit z_i of the form $z_i \in \left\{\overline{a}, \overline{a-1}, \ldots, \overline{1}, 0, 1, \ldots, a-1, a\right\}$, where $5 \leq a \leq 9$. In DSD arithmetic, a DSD number Z expressed as $Z = z_{n-1}z_{n-2}\ldots z_1 z_0 = \sum_{i=0}^{n-1} z_i \times 10^i$ is an n-digit array with DSD z_i in a maximally redundant set

$$\left\{\overline{9}, \overline{8}, \ldots, \overline{1}, 0, 1, \ldots, 8, 9\right\}. \quad (4)$$

Although a DSD can be represented in 5 bits in 2's complement format [28], in this paper, the 8-bit format is used because it is a natural extension to the traditional binary signed-digit (BSD) format. Therefore, it is quite possible that this representation makes building circuits capable of performing both binary and decimal division more feasible. In addition, as explained later, while an 8-bit DSD number can be negated using an array of inverters, this needs a 2's complement operation in the 5-bit DSD representation.

The DSD z_i is represented by a BSD vector as $z_i = z_{i3}z_{i2}z_{i1}z_{i0}$, where $z_{ik} = \left(z_{ik}^+, z_{ik}^-\right) \in \left\{\overline{1}, 0, 1\right\}$ and $k = 0,1,2,3$. Hence, each DSD is encoded as 8 bits. The value of Z can be determined as $Z = (Z^+, Z^-) = z_{n-1}^+ z_{n-2}^+ \ldots z_1^+ z_0^+ - z_{n-1}^- z_{n-2}^- \ldots z_1^- z_0^-$, where the operator "$-$" refers to decimal subtraction. A BCD to DSD format conversion can be performed in zero time with no use of hardware. However, converting from DSD to BCD, like all conversions from redundant to nonredundant formats, requires a time consuming operation.

A DSD number $Z = (Z^+, Z^-)$ can be negated as

$$-Z = \left(Z^-, Z^+\right) = \left(\neg Z^+, \neg Z^-\right) \tag{5}$$

where "$\neg$" denotes a 1's complement (invert) function. For example, while number 9 can be represented in DSD as (1001,0000), using (5), -9 can be expressed as either $(0000, 1001) = \overline{1}00\overline{1}$ or $(0110, 1111) = \overline{1}00\overline{1}$ since in BSD arithmetic $1 = (1, 0)$, 0= either (1,1) or (0,0) and $\overline{1}$ = (0,1).

A new DCF addition is presented in [16]. This addition, like its binary predecessors [29]–[31], limits carry propagation to a small number of digit positions to the left and, therefore, all digitwise addition operations, irrespective of their length, execute in the same time. The adder can be easily simplified if the addends are either two BCD numbers or one BCD and one DSD number.

An interesting feature of DSD arithmetic is that subtraction $W = U - V$, where U and V are two n-digit BCD numbers, can be performed with no hardware time delay as $W = (W^+, W^-) = (U, V)$, where W is in the DSD format.

It should be noted that in the remainder of the paper, the word "digit" is frequently used. Where this is not further qualified, based on the context, "digit" refers to a decimal digit, represented in either DSD or BCD.

V. DFP in IEEE 754R Standard Proposal

In the proposed revision to the IEEE 754 standard, the encodings for decimal numbers allow for a range of positive and negative values together with values of ± 0, $\pm\infty$, and not-a-number (NaN) [32]. Three formats of decimal numbers are allowed as follows:

- *decimal32* numbers, which are encoded in four consecutive bytes (32 bits);
- *decimal64* numbers, which encoded in eight consecutive bytes (64 bits);
- *decimal128* numbers, which are encoded in 16 consecutive bytes (128 bits).

A finite DFP number is defined by a *sign*, an *exponent*, and a decimal integer *coefficient*. The value of the DFP number F is given by $F = (-1)^{sign} \times \text{coefficient} \times 10^{exponent}$. In this representation, *sign* is a single bit (as in IEEE 754 standard for binary FP), exponent is encoded as an unsigned binary integer from which a $bias$ is subtracted, and *coefficient* is an unsigned decimal integer. The three DFP formats, decimal32, decimal64, and decimal128, have 7-, 16-, and 34-B *coefficients*, respectively. The representation format proposed for *coefficient* uses densely packed decimal encoding [33]. It is a compressed form of the traditional binary-coded decimal (BCD) format. This encoding is a lossless algorithm, which compresses three BCD digits into 10 bits. The algorithm can be applied or reversed using only simple Boolean operations.

Unlike the binary FP representation in which the significand is a normalized number, *coefficient* has no such limitation. This means that it can take any value between 0 and $10^{clength} - 1$, where $clength$ is the length of *coefficient* in decimal digits.

A. Precision

Precision p, is a positive integer, which sets the maximum number of significant digits that can result from an arithmetic operation. The upper limit to it can be the length of the *coefficient* supported by the decimal representation format, i.e., 7, 16, or 34 digits corresponding to decimal32, decimal64, and decimal128, respectively. There is a lower limit on the setting p, which may be the same as the upper limit.

B. Rounding

According to the proposal of the IEEE 754R standard [8], a final quotient q that might be represented in more digits than the standard requires should be rounded to fit the destination format. Among different rounding modes introduced by the proposal, round to nearest even (RNE) is the mode used in the decimal division developed in this paper since it rounds with zero bias[2] [22].

VI. DFP Division Using SRT Algorithm

This section introduces DFP division using the high-radix SRT algorithm, i.e., $r = 10$. The proposed method fulfills the requirements of the proposal of the IEEE 754R standard [8].

Assumptions

- The radix r is equal to 10.
- It is known that in high-radix SRT division, to perform the recurrence using a carry-free adder (CFA) [22] in radix r, the SD set from which the quotient digit is selected must satisfy $\lceil r + 1/2 \rceil \leq a \leq r - 1$ [22]. Therefore, for DFP division, $6 \leq a \leq 9$, where a is the largest allowed value for the quotient digit. Investigation shows that almost the same propagation delay occurs when generating either set of the multiples of d used in the SRT recurrence of DFP division: $\{d, 2d, \ldots, 5d, 6d\}$ or $\{d, 2d, \ldots, 8d, 9d\}$. So, for the DFP division described here

$$q_{j+1} \in \left\{\overline{9}, \overline{8}, \ldots, \overline{1}, 0, 1, \ldots, 8, 9\right\} \tag{6}$$

which is the maximally redundant set corresponding to $\rho = 1$. This choice makes the QDS function simpler and, therefore, reduces its delay.

[2]bias is the average rounding error.

A. DFP Division Formulation

Considering w as the PR and p as the precision, the DFP division definition (for nonzero and nonspecial operands)[3] can be expressed as follows.

- The unpacked (BCD) representations of the dividend *coefficient*, $coefficient_X = x_{p-1}x_{p-2}\ldots x_1x_0$, and the divisor *coefficient*, $coefficient_D = d_{p-1}d_{p-2}\ldots d_1d_0$, are written in the fraction form as $f_X = 0.x_{p-1}x_{p-2}\ldots x_1x_0$ and $f_D = 0.d_{p-1}d_{p-2}\ldots d_1d_0$, respectively.
- With the appropriate number of left shifts, the fractions are normalized such that

$$\frac{1}{10} \leq x < 1 \quad \text{and} \quad \frac{1}{10} \leq d < 1 \tag{7}$$

  where x and d represent normalized f_X and f_D, respectively. Consequently, the dividend exponent, $exponent_X$, and the divisor exponent, $exponent_D$, should be modified accordingly. Using the new values, the quotient exponent is set as

$$exponent_Q = -p + (exponent_X - exponent_D). \tag{8}$$

- The decimal QDS function selects the correct value for the quotient digit q_{j+1} from DSD set (6) so that the convergence condition

$$-d \leq w[j+1] < d \tag{9}$$

  is always satisfied.
- The decimal SRT recurrence

$$w[j+1] = 10w[j] - q_{j+1}d \tag{10}$$

  where $j = 0, 1, \ldots, p$ is used to generate the next PR. Since the inequality $x < d$ cannot always be guaranteed, in order to satisfy the convergence condition (9), the PR is initialized as

$$w[0] = \frac{x}{10}. \tag{11}$$

  The PR is represented in the DSD format and, therefore, (10) can be carried out using DCF addition [16].
- Using the quotient digits generated in every recurrence, q is formed as

$$q = \sum_{j=0}^{p} q_{j+1}10^{p-(j+1)} = q_1q_2\ldots q_{p-1}q_p \overset{\substack{\text{decimal point}\\ \downarrow}}{.} q_{p+1} \tag{12}$$

  after $p+1$ cycles. The additional digit q_{p+1} is used later for rounding.
- The value of $sign_Q$ is set as

$$sign_Q = sign_X \text{ XOR } sign_D. \tag{13}$$

B. Convert and Round

In the DFP divider, RNE is considered as the default rounding mode. As in binary dividers, after the $(p+1)$th division iteration, another cycle is needed to complete the quotient conversion and rounding processes. In that cycle, the last PR, $w[p+1]$, is *sign* detected by the convert and round (CR) unit as

$$S_{p+1} = \begin{cases} 0, & \text{if } w[p+1] \geq 0 \text{ and} \\ 1, & \text{if } w[p+1] < 0. \end{cases} \tag{14}$$

Unlike binary rounding, the decimal rounding may encounter a halfway condition [34]. Examples include

$$\frac{0.10}{0.16} = 0.625 \qquad \frac{0.12}{0.32} = 0.375. \tag{15}$$

To deal with this, the final PR is zero detected in the $(p+2)$th cycle as

$$Z_{p+1} = \begin{cases} 0, & \text{if } w[p+1] \neq 0 \\ 1, & \text{if } w[p+1] = 0. \end{cases} \tag{16}$$

Then, the rounded q is determined using the values formed by the on-the-fly rounding algorithm [23] in the $QM[p] = Q[p]-1$, $Q[p]$, and $QP[p] = Q[p]+1$ registers, where 1 refers to 1 unit of the last position (*ulp*), and the rules shown in Table I. It should be noted that the final q is not post-normalized because the decimal representation in the proposal of the IEEE 754R standard is a nonnormalized format.

TABLE I
RULES USED BY THE DECIMAL CR UNITS TO REPRESENT THE UNROUNDED QUOTIENT IN THE PROPOSAL OF THE IEEE 754R STANDARD FORMAT. S AND Z ARE DEFINED BY (14) AND (16), RESPECTIVELY

	q rounded to p digits		
	(S_{p+1}, Z_{p+1})		
q_{p+1}	$(1,\times)$	$(0,0)$	$(\times,1)$
$\bar{9}$	$QM[p]$	$QM[p]$	$QM[p]$
$\bar{8}$	$QM[p]$	$QM[p]$	$QM[p]$
$\bar{7}$	$QM[p]$	$QM[p]$	$QM[p]$
$\bar{6}$	$QM[p]$	$QM[p]$	$QM[p]$
$\bar{5}$	$QM[p]$	$Q[p]$	$QM[p]$ if $QM[p]\langle lsb\rangle = 0$; $Q[p]$ if $QM[p]\langle lsb\rangle = 1$
$\bar{4}$	$Q[p]$	$Q[p]$	$Q[p]$
$\bar{3}$	$Q[p]$	$Q[p]$	$Q[p]$
$\bar{2}$	$Q[p]$	$Q[p]$	$Q[p]$
$\bar{1}$	$Q[p]$	$Q[p]$	$Q[p]$
0	$Q[p]$	$Q[p]$	$Q[p]$
1	$Q[p]$	$Q[p]$	$Q[p]$
2	$Q[p]$	$Q[p]$	$Q[p]$
3	$Q[p]$	$Q[p]$	$Q[p]$
4	$Q[p]$	$Q[p]$	$Q[p]$
5	$Q[p]$	$QP[p]$	$Q[p]$ if $Q[p]\langle lsb\rangle = 0$; $QP[p]$ if $Q[p]\langle lsb\rangle = 1$
6	$QP[p]$	$QP[p]$	$QP[p]$
7	$QP[p]$	$QP[p]$	$QP[p]$
8	$QP[p]$	$QP[p]$	$QP[p]$
9	$QP[p]$	$QP[p]$	$QP[p]$

Note: $\langle lsb\rangle$ is the least significant bit of the number's least significant digit.

C. Dealing With Exact Results

An issue unique to DFP division is to represent exact quotients correctly. Based on the proposal of the IEEE 754R standard, if the final quotient is exact then decimal division should return the *coefficient* and the *exponent* of the correct value. The *exponent* has to be the closest value to the ideal *exponent*. The

[3]This simplifying assumption prevents any exception handling.

ideal *exponent* is the original dividend *exponent* minus the original divisor *exponent*. In our implementation of DFP division, $zero_j$, which is calculated every iteration as

$$Z_j = \begin{cases} 0, & \text{if } w[j] \neq 0 \\ 1, & \text{if } w[j] = 0 \end{cases} \tag{17}$$

signals whether the quotient is exact. Once the jth PR is found equal to zero, one can conclude that all the consequent quotient digits are zero and, therefore, the division is exact. In this case, q can be adjusted by an appropriate number of right or left shifts. Correspondingly, using the value of j, exponent_Q obtained from (8) should be adjusted. The circuit performing this operation can be embedded in the CR unit.

D. Decimal QDS Function

This section explains how the comparison multiples method [18] has been used to develop the QDS function of the DFP divider. In this method, the quotient digit's magnitude is obtained by comparison of the truncated PR with truncated multiples of d. Meanwhile, another circuit determines the sign of the quotient digit by checking only a few most significant digits of the PR.

Having considered the assumption given in Section VI-A and the normalizations (7), substituting (1) in (3) and adding dq_{j+1} results in

$$d(q_{j+1} - 1) \leq 10w[j] \leq d(q_{j+1} + 1). \tag{18}$$

Considering (3), since $q_{j+1} = k$ can take any of the 19 values in $\{\overline{9}, \overline{8}, \ldots, \overline{1}, 0, \;1, \ldots, 8, 9\}$, the range (18) can be sliced into 19 intervals in the general form of $d(k-1) \leq 10w[j] \leq d(k+1)$. Each interval is associated with a member of the SD set. So, to find an appropriate value for q_{j+1}, it is sufficient to investigate which interval contains $10w[j]$. This means that the QDS function can be expressed alternatively as

$$q_{j+1} = k \in \{\overline{9}, \overline{8}, \ldots, \overline{1}, 0, 1, \ldots, 8, 9\}, \quad \text{if } d(k-1) \leq 10w[j] \leq d(k+1). \tag{19}$$

Table II shows the QDS function (19) in an expanded form. Since (3) must always be satisfied and also because there is no possible choice for q_{j+1} larger (smaller) than 9 ($\overline{9}$), interval $8d \leq 10w[j] \leq 10d \, (-10d \leq 10w[j] \leq -8d)$ is replaced by $8d \leq 10w[j] \, (10w[j] \leq -8d)$ in the table.

Investigation reveals that the intervals in Table II always have some overlaps, where there are two choices for the q_{j+1}. Therefore, to make the QDS function one-to-one, every two conjunct intervals are detached using separating points, namely the comparison multiples. The comparison multiple M_k is chosen such that

$$d(k-1) \leq M_k \leq dk \tag{20}$$

where $k \in \{\overline{9}, \overline{8}, \ldots, \overline{1}, 0, 1, \ldots, 8, 9\}$. The radix-$r$ nonredundant number M_k is represented as $M_k = A_k d$, where A_k is a rational number. The QDS function shown in Table II can be expressed as

TABLE II
DIFFERENT EXPRESSION OF THE QDS FUNCTION

q_{j+1}	Condition
$\overline{9}$	$10w[j] \leq -8d$
$\overline{8}$	$-9d \leq 10w[j] \leq -7d$
$\vdots$	$\vdots$
$\overline{1}$	$-2d \leq 10w[j] \leq 0$
0	$-d \leq 10w[j] \leq d$
1	$0 \leq 10w[j] \leq 2d$
$\vdots$	$\vdots$
8	$7d \leq 10w[j] \leq 9d$
9	$8d \leq 10w[j]$

$$q_{j+1} = \begin{cases} \overline{9}, & \text{if } 10w[j] < M_{-8} \\ \overline{8}, & \text{if } M_{-8} \leq 10w[j] < M_{-7} \\ \vdots & \\ 0, & \text{if } M_0 \leq 10w[j] < M_1 \\ \vdots & \\ 8, & \text{if } M_8 \leq 10w[j] < M_9 \\ 9, & \text{if } M_9 \leq 10w[j]. \end{cases} \tag{21}$$

Fortunately, existence of the overlaps means that there are sometimes more than one possible value for q_{j+1}. This allows the QDS function to compare just the most significant c fractional digits of $10w[j]$ with M_k truncated to c fractional bits. So, (21) can be rewritten into the truncated form

$$q_{j+1} = \begin{cases} \overline{9}, & \text{if } \{10w[j]\}_c < \{M_{-8}\}_c \\ \overline{8}, & \text{if } \{M_{-8}\}_c \leq \{10w[j]\}_c < \{M_{-7}\}_c \\ \vdots & \\ 0, & \text{if } \{M_0\}_c \leq \{10w[j]\}_c < \{M_1\}_c \\ \vdots & \\ 8, & \text{if } \{M_8\}_c \leq \{10w[j]\}_c < \{M_9\}_c \\ 9, & \text{if } \{M_9\}_c \leq \{10w[j]\}_c \end{cases} \tag{22}$$

which requires 18 comparisons

$$\{10w[j]\}_c - \{M_k\}_c \tag{23}$$

followed by 18 sign detections and a coder to be implemented.

E. Optimized QDS Function

The two major ideas to speed up the QDS function, and consequently, the recurrence cycle time, are as follows:

1) breaking the critical path into two or more concurrent, but shorter paths;
2) decreasing the fan-out of the circuits delivering $\{10w[j]\}_c$ to the QDS function.

From (20) and Table II, a symmetry among the margins can be found. Therefore, having used the substitutions

$$\{M_{-k+1}\}_c = \{-M_k\}_c = -\{M_k\}_c. \tag{24}$$

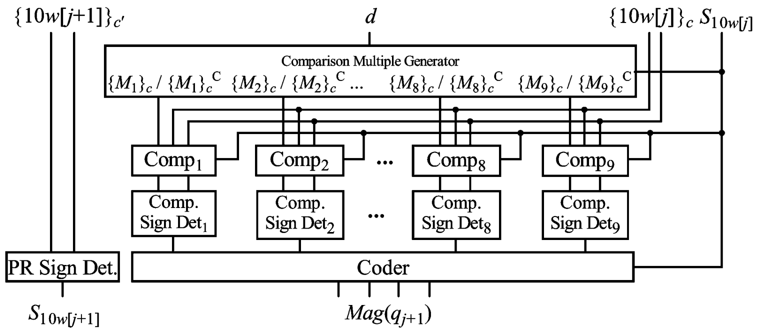

Fig. 1. Comparison multiples based QDS function used in DFP division. Superscript C denotes a 9's complement. Subscripts c and c' are defined in Section VI-H.

Equation (22) is redefined as

$$q_{j+1}=\begin{cases}\overline{9}, & \text{if } 10w[j]<0 \wedge \{10w[j]\}_c<-\{M_9\}_c\\ \overline{8}, & \text{if } 10w[j]<0 \wedge -\{M_9\}_c\le\{10w[j]\}_c<-\{M_8\}_c\\ \vdots & \\ \overline{1}, & \text{if } 10w[j]<0 \wedge -\{M_2\}_c\le\{10w[j]\}_c<-\{M_1\}_c\\ 0, & \text{if } 10w[j]<0 \wedge -\{M_1\}_c\le\{10w[j]\}_c\\ 0, & \text{if } 10w[j]\ge 0 \wedge \{10w[j]\}_c<\{M_1\}_c\\ 1, & \text{if } 10w[j]\ge 0 \wedge \{M_1\}_c\le\{10w[j]\}_c<\{M_2\}_c\\ \vdots & \\ 8, & \text{if } 10w[j]\ge 0 \wedge \{M_8\}_c\le\{10w[j]\}_c<\{M_9\}_c\\ 9, & \text{if } 10w[j]\ge 0 \wedge \{M_9\}_c\le\{10w[j]\}_c\end{cases} \tag{25}$$

where "$\wedge$" is logical "AND" and $k \in \{1,\ldots,a\}$. The number of comparisons in (25) decreases to 9; however, (25) cannot be fulfilled unless the sign of the shifted PR is known. Checking the sign of $10w[j]$, while represented in the DSD format, needs a full-length carry generation. This causes an impracticable response time proportional to $\log_2$ of the operand width. Investigation in Section VI-H shows that existence of the overlaps helps to convert the full-length sign detection to a limited-range. In other words, to make the decision whether $\{M_k\}_c$ or $-\{M_k\}_c$ should participate in the comparisons, it is not necessary to know the exact sign of $10w[j]$, but the sign of $10w[j]$ truncated to c' fractional digits. So, (25) changes to (26), shown at the bottom of the page. Fig. 1 shows the general structure of the QDS function (26). The design's subunits are explained later.

F. DFP Recurrence

Fig. 2 shows an implementation for the recurrence of DFP division using the QDS function shown in Fig. 1. For simplicity, the CR unit is not shown. This design follows the general structure of the high-radix SRT recurrence [15], [34]; however, due to the special features of the proposed QDS function, minor changes are applied. In the following, the subunits involved in the DFP recurrence are briefly introduced.

1) PR Formation: In Fig. 2, the PR formation unit calculates all the possible candidates for the next PR, $w[j+1]$. They are named w_k corresponding to the values obtained from $10w[j] - kd$, where $k = 0, 1, \ldots, 8, 9$. These values are kept in registers and in the next iteration, the correct value for the PR is selected using the magnitude of the quotient digit and MUX 11:1.

Considering 1-digit normalizations (7) and the fact that $10w[j]$ has one integer digit, the PR formation can be constructed using a $(p+2)$-digit DCF adder [16] with a DSD addend and a BCD augend. Representation overflow may increase the length of w_k to $p+2$. A small adjust circuit can be developed based on the following observation to cancel the overflow every iteration [18]. Since the convergence condition (9) is always true, regardless of the exact value of d, the 3 most significant digits of the next PR can take one of the combinations $00.a$, $00.\overline{a}$, $01.\overline{b}$, $0\overline{1}.b$, $1\overline{9}.\overline{b}$, and $\overline{1}9.b$, where $a \in \{0, 1, \ldots, 8, 9\}$ and $b \in \{1, 2, \ldots, 8, 9\}$.

2) Factor Generator: The factor generator used in the DFP divider provides the multiples and the 9's complemented multiples of d not only in full range, but also in truncated form to supply the comparison multiple generator.

$$q_{j+1}=\begin{cases}\overline{9}, & \text{if } \{10w[j]\}_{c'}<0 \wedge \{10w[j]\}_c<-\{M_9\}_c\\ \overline{8}, & \text{if } \{10w[j]\}_{c'}<0 \wedge -\{M_9\}_c\le\{10w[j]\}_c<-\{M_8\}_c\\ \vdots & \\ \overline{1}, & \text{if } \{10w[j]\}_{c'}<0 \wedge -\{M_2\}_c\le\{10w[j]\}_c<-\{M_1\}_c\\ 0, & \text{if } \{10w[j]\}_{c'}<0 \wedge -\{M_1\}_c\le\{10w[j]\}_c\\ 0, & \text{if } \{10w[j]\}_{c'}\ge 0 \wedge \{10w[j]\}_c<\{M_1\}_c\\ 1, & \text{if } \{10w[j]\}_{c'}\ge 0 \wedge \{M_1\}_c\le\{10w[j]\}_c<\{M_2\}_c\\ \vdots & \\ 8, & \text{if } \{10w[j]\}_{c'}\ge 0 \wedge \{M_8\}_c\le\{10w[j]\}_c<\{M_9\}_c\\ 9, & \text{if } \{10w[j]\}_{c'}\ge 0 \wedge \{M_9\}_c\le\{10w[j]\}_c\end{cases} \tag{26}$$

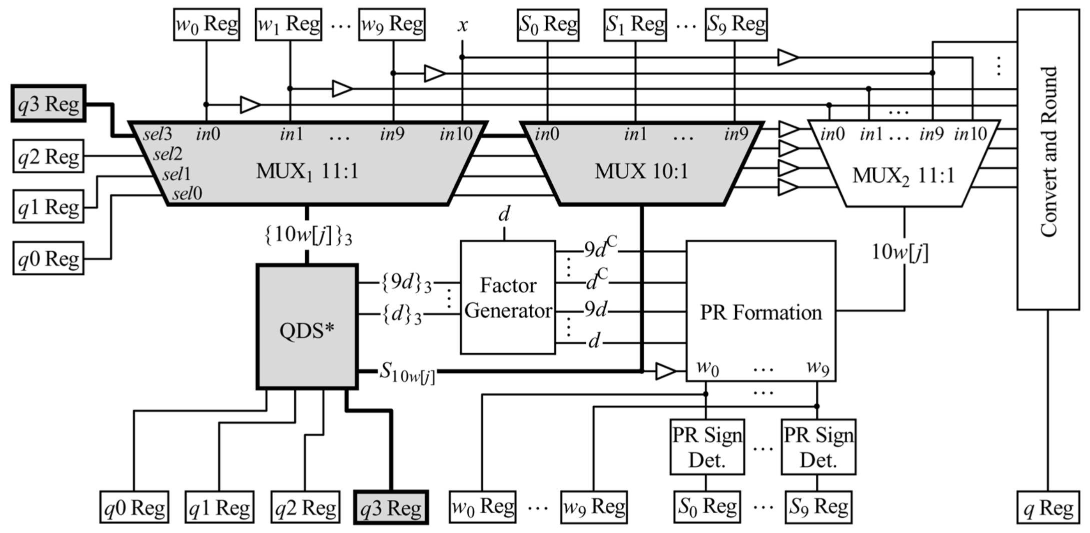

Fig. 2. Recurrence of the proposed DFP division. QDS^* is the QDS function shown in Fig. 1 without the PR sign detector. The critical path is shown in gray. Superscript C denotes a 9's complement.

Using the following scheme, the appropriate multiples of d are produced in full-range in the BCD format as quickly as possible.

1) Having produced $10d$ in the BCD format just by a wired left shift, value $9d$ is calculated in the DSD format as $9d = 10d - d$. Meanwhile, $2d$ is produced in the DSD form as $2d = d + d$.
2) Then, using the values obtained, $3d$, $4d$, and $8d$ are calculated in the DSD format as $3d = 2d + d$, $4d = 2d + 2d$, and $8d = 9d - d$.
3) Afterward, $5d$, $6d$, and $7d$ are generated and represented in DSD as $5d = 8d - 3d$, $6d = 8d - 2d$, and $7d = 9d - 2d$.
4) The multiples of d represented in the DSD format are reformatted into the BCD form.

In this scheme, subtractions with BCD minuend and subtrahend are performed as explained in Section IV. The other addition and subtraction operations can be carried out using DCF addition [16]. DSD to BCD conversion can be implemented by appropriately modifying any BSD to 2's complement conversion approaches [35], [36] with parallel-prefix carry generating networks [37].

Due to the complex and time consuming operation performed by the factor generator, it needs an interval equal to one recurrence cycle time to generate kd and kd^{C} in the BCD format. With this initializing cycle at the beginning of DFP division (i.e., before the first recurrence iteration starts), DFP division produces the final quotient in $p + 3$ cycles.[4]

G. Precisions of QDS Function Operands

1) Determining c and e: To determine c, which denotes the number of fractional digits of $10w[j]$ (as well as the number of fractional bits of M_k) involved in the DCF additions, the comparison intervals shown in (26) are studied. For simplicity, only the general interval shown as $\{M_k\}_c \le \{10w[j]\}_c < \{M_{k+1}\}_c$ is investigated. However, the other cases can be derived in the same way. The interval is divided into

$$\{M_k\}_c \le \{10w[j]\}_c \tag{27a}$$

$$\{10w[j]\}_c < \{M_{k+1}\}_c. \tag{27b}$$

It is known that if BCD number Y is truncated to t digits of precision right of the decimal point, then

$$\{Y\}_t \le Y < \{Y\}_t + 10^{-t}. \tag{28}$$

However, for DSD number Z, the same truncation results in

$$\{Z\}_t - 10^{-t} < Z < \{Z\}_t + 10^{-t}. \tag{29}$$

Using (28) and (29), (27a) changes to $M_k - 10^{-c} < \{M_k\}_c \le \{10w[j]\}_c < 10w[j] + 10^{-c}$ or simply

$$M_k - 10^{-c+1} < 10w[j]. \tag{30}$$

Since $q_{j+1} = k$, adding $-kd$ to (30) and using (1) results in $M_k - 10^{-c+1} - kd < w[j+1]$. To maintain the convergence condition (3), the inequality $-d \le M_k - 10^{-c+1} - kd$ or equivalently

$$d(k-1) + 10^{-c+1} \le M_k \tag{31}$$

must be complied with. For interval (27b), a similar derivation can be used to obtain

$$M_{k+1} \le d(k+1) - 10^{-c}. \tag{32}$$

Replacing $k+1$ with k in (32) and combining with (31) gives

$$d(k-1) + 10^{-c+1} \le M_k \le dk - 10^{-c}. \tag{33}$$

This inequality gives tighter ranges than condition (20). This is because rather than full-range, truncated comparison multiples are used in the comparisons. To make sure that finding a value for M_k using (33) is always possible, the inequality $d(k-1)+$

[4]One initialization cycle, $p + 1$ iterations to generate $p + 1$ quotient digits and one cycle for rounding.

$10^{-c+1} < dk - 10^{-c}$ should always be maintained. This leads to $10^c > 11/d$. Since $d \geq 1/10$, it follows that $10^c > 110$ or

$$c \geq 3 \tag{34}$$

which gives the lower bound on c for the comparison multiples-based QDS function used in DFP division.

In addition to c, it is required to determine e, which is the width of the integer section of the operands involved in the comparisons (23). Having considered M_9 as the largest comparison multiple and $d < 1$, inequality (20) results in $M_9 < 9$. This means that M_9 is a BCD number with at most one integer digit. On the other hand, Fig. 2 indicates that the comparators receive $10w[j]$, which has one digit in its integer section. These two observations lead to $e = 1$. It should be noted that the representation overflow [38] may make the inputs to the comparison sign detectors one digit wider.

2) Determining c' and e': In Fig. 1, the PR sign detector finds the polarity of $\{10w[j+1]\}_{c'}$ represented as a $(e'+c')$-digit SD number. Since the more digits the PR sign detector checks, the larger its response time is, it is useful to derive the lower bounds on e' and c'.

From (29), it can be derived that $0 \leq \{10w[j]\}_{c'} < 10w[j] + 10^{-c'}$ or equivalently $-10^{-c'} < 10w[j]$. To make this inequality comply with the convergence condition (3) in the neighborhood of 0 with $q_{j+1} = 0$, the inequality $10^{-c'} \leq d$ must be satisfied. Since $d \geq 1/10$, this inequality can be changed to a tighter condition independent to d as $10^{c'} \geq 10$ or

$$c' \geq 1. \tag{35}$$

To find e', one can see that since in Fig. 2 the w_k registers store numbers with no integer digits, $10w[j]$ (the output of MUX 11:1), can have just one integer digit. Therefore, when $10w[j]$ is applied to the PR formation, $w_0[j+1]$ to $w_9[j+1]$, which are the candidates for the next PR ($w[j+1]$), will have at most two integer digits. This means that for $10w[j+1]$, which is used by the PR sign detector in Fig. 1, $e' = 3$.

H. QDS Function Implementation

Having assumed $c+e = 4$ and $c'+e' = 4$, the main modules involved in the implementation are explained as follows.

1) Limited-Range Comparators: For every $k \in \{1, 2, \ldots, 9\}$, a four-digit DCF adder performs

$$\{10w[j]\}_3 - \{M_k\}_3, \quad \text{if } \{10w[j]\}_1 \geq 0 \tag{36a}$$
$$\{10w[j]\}_3 + \{M_k\}_3, \quad \text{if } \{10w[j]\}_1 < 0 \tag{36b}$$

or equivalently

$$\{10w[j]\}_3 + \{M_k\}_3^{\mathrm{C}} + \neg S_{10w[j]}, \quad \text{if } S_{10w[j]} = 0 \tag{37a}$$
$$\{10w[j]\}_3 + \{M_k\}_3 + \neg S_{10w[j]}, \quad \text{if } S_{10w[j]} = 1 \tag{37b}$$

where Y^{C} is 9's complement of Y and

$$S_{10w[j]} = \begin{cases} 0, & \text{if } \{10w[j]\}_1 \geq 0 \\ 1, & \text{if } \{10w[j]\}_1 < 0. \end{cases} \tag{38}$$

In Fig. 1, each comparator "Comp_n" can be realized using a DCF adder introduced by Nikmehr *et al.* [16] simplified for a DSD addend and a BCD augend.

2) Limited-Range Comparison Sign Detectors: Unlike BCD numbers, the sign of a DSD number is equal to the sign of its most significant nonzero digit. It means that a DSD sign detection may require an investigation over the length of the input operand. Each comparator in the QDS function is followed by a five-digit sign detector shown as "Comp Sign Det_n" in Fig. 1. To find the signs of the DSD results obtained from (37), there are nine sign detectors in the system. They can be implemented using networks similar to those used for precalculating the input carries in either parallel-prefix adders [37] or reverse-carry adders [39]. Therefore, having considered the fact that the input operands are very short, a comparison sign detector does not incur large delay. The sign bits are later used to form the magnitude of the quotient digit.

3) Limited-Range PR Sign Detector: The comparisons (37) cannot be accomplished unless the sign of $\{10w[j]\}_1$ is already known. Therefore, to prevent any additional delay to the iteration response time, the QDS function should be implemented in such a way that the sign of $\{10w[j]\}_1$ becomes available before (37) starts. Since $x > 0$, the sign of $\{10w[0]\}_1$ is already known before the first iteration begins. Therefore, selecting a value for q_1 using (26) is independent of PR sign detection in the first iteration. This observation can be extended to the jth iteration to find the polarity of $\{10w[j+1]\}_1$.

The PR sign detector is based on (38). This operation is performed in parallel to the rest of the QDS function and, therefore, does not affect the iteration delay.

The four-digit PR sign detector, which is shown as "PR Sign Det" in Fig. 1, is implemented in the same way as the comparison sign detectors; however, as shown in Fig. 2, when the QDS function is used in the DFP recurrence, it is duplicated in ten copies and put after the PR formation. According to this structure, the PR sign detectors find the polarity of $10w_k[j+1]$, for $k = 0, 1, \ldots, 8, 9$. These signs are stored in the S_k registers and then in the next iteration, one of them is selected as the correct sign using the magnitude of the quotient digit and MUX 10:1.

4) Coder: In the proposed QDS function for DFP division, the DSD quotient digit q_{j+1} is represented in the sign-magnitude format using $Sign(q_{j+1})$ and $Mag(q_{j+1})$. In this representation

$$Sign(q_{j+1}) = \begin{cases} \text{don't care}, & \text{if } q_{j+1} = 0 \\ S_{10w[j]}, & \text{otherwise} \end{cases} \tag{39}$$

determines the sign and $Mag(q_{j+1})$ indicates the magnitude (absolute) of the value selected for q_{j+1}. In other words,

$$Mag(-q_{j+1}) = Mag(q_{j+1}). \tag{40}$$

Based on the value of $\{10w[j]\}_3$, the comparison sign detectors produce nine bits, namely S_1 to S_9. These bits, along with $Sign(q_{j+1}) = S_{10w[j]}$, are used by the coder to construct $Mag(q_{j+1})$. The values represented by S_1 to S_9 as well as their relationship with the exact value of q_{j+1} are shown in Table III. Investigating the table reveals that $Mag(q_{j+1}) = q_3q_2q_1q_0$ can

TABLE III
FORMING q_{j+1} USING $Sign(q_{j+1})$ AND $Mag(q_{j+1}) = q3q2q1q0$. $Sign$ AND Mag ARE DEFINED BY (39) AND (40), RESPECTIVELY

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	$Sign(q_{j+1})$	$q3q2q1q0$	q_{j+1}
1	1	1	1	1	1	1	1	1	1	1111	$\bar{9}$
1	1	1	1	1	1	1	1	0	1	1110	$\bar{8}$
1	1	1	1	1	1	1	0	0	1	1101	$\bar{7}$
1	1	1	1	1	1	0	0	0	1	1100	$\bar{6}$
1	1	1	1	1	0	0	0	0	1	1011	$\bar{5}$
1	1	1	1	0	0	0	0	0	1	1010	$\bar{4}$
1	1	1	0	0	0	0	0	0	1	1001	$\bar{3}$
1	1	0	0	0	0	0	0	0	1	1000	$\bar{2}$
1	0	0	0	0	0	0	0	0	1	0111	$\bar{1}$
0	0	0	0	0	0	0	0	0	1	0110	0
1	1	1	1	1	1	1	1	1	0	0110	0
0	1	1	1	1	1	1	1	1	0	0111	1
0	0	1	1	1	1	1	1	1	0	1000	2
0	0	0	1	1	1	1	1	1	0	1001	3
0	0	0	0	1	1	1	1	1	0	1010	4
0	0	0	0	0	1	1	1	1	0	1011	5
0	0	0	0	0	0	1	1	1	0	1100	6
0	0	0	0	0	0	0	1	1	0	1101	7
0	0	0	0	0	0	0	0	1	0	1110	8
0	0	0	0	0	0	0	0	0	0	1111	9

TABLE IV
VALID RANGES FOR M_k

k	Range	A_k	M_k
1	$\frac{1}{100} \leq M_1 \leq d - \frac{1}{1000}$	$\frac{5}{10}$	$\frac{5}{10}d$
2	$d + \frac{1}{100} \leq M_2 \leq 2d - \frac{1}{1000}$	$\frac{15}{10}$	$\frac{15}{10}d$
3	$2d + \frac{1}{100} \leq M_3 \leq 3d - \frac{1}{1000}$	$\frac{25}{10}$	$\frac{25}{10}d$
4	$3d + \frac{1}{100} \leq M_4 \leq 4d - \frac{1}{1000}$	$\frac{35}{10}$	$\frac{35}{10}d$
5	$4d + \frac{1}{100} \leq M_5 \leq 5d - \frac{1}{1000}$	$\frac{45}{10}$	$\frac{45}{10}d$
6	$5d + \frac{1}{100} \leq M_6 \leq 6d - \frac{1}{1000}$	$\frac{55}{10}$	$\frac{55}{10}d$
7	$6d + \frac{1}{100} \leq M_7 \leq 7d - \frac{1}{1000}$	$\frac{65}{10}$	$\frac{65}{10}d$
8	$7d + \frac{1}{100} \leq M_8 \leq 8d - \frac{1}{1000}$	$\frac{75}{10}$	$\frac{75}{10}d$
9	$8d + \frac{1}{100} \leq M_9 \leq 9d - \frac{1}{1000}$	$\frac{85}{10}$	$\frac{85}{10}d$

be generated using S_1 to S_9 and $Sign(q_{j+1})$ by implementing expressions

$$q3 = S'_8 \tag{41a}$$

$$q2 = S'_4 \vee \neg S'_8 \tag{41b}$$

$$q1 = S'_2 \vee (S'_6 \wedge \neg S'_4) \vee (\neg S'_8 \wedge \neg S'_7) \tag{41c}$$

$$q0 = S'_1 \vee (S'_3 \wedge \neg S'_2) \vee (S'_5 \wedge \neg S'_4) \vee (S'_7 \wedge \neg S'_6) \vee (S'_9 \wedge \neg S'_8) \tag{41d}$$

where $S'_k = S_k \,\text{XNOR}\, Sign(q_{j+1})$ for $k = 1, 2, \ldots, 9$ and "$\vee$" is logical "OR."

5) Comparison Multiple Generator: To perform the comparisons (37), nine positive comparison multiples $\{M_k\}_3$ and nine 9's complement comparison multiples $\{M_k\}_3^C$, where $k \in \{1, 2, \ldots, 8, 9\}$, are provided by the comparison multiple generator. The generator produces $\{M_k\}_3$ and then, using these values, $\{M_k\}_3^C$ are supplied.

From (33), M_k can be selected in the range $d(k-1) + (1/100) \leq M_k \leq kd - (1/1000)$ for $k = 1, 2, \ldots, 9$. The ranges are listed in Table IV in detail. In the rightmost column of the table, one set of the most easily-calculated values for M_k is shown. Having assumed that the integer multiples of d, kd, are already available in the BCD format, $\{M_1\}_3$ can be obtained just by truncating $5d$ to two fractional digits and then shifting the result one digit to the right as $\{M_1\}_3 = \{5d\}_2/10$. However, to calculate the rest of the comparison multiples, the calculation

$$\{M_k\}_3 = \frac{\{10kd - 5d\}_2}{10} \tag{42}$$

for $k = 2, \ldots, 8, 9$ is required. For a given k, (42) is performed as a full length BCD subtraction $10kd - 5d$ followed by a two-digit truncation over the result and a one-digit right shift. As shown, the full length subtraction stage incurs an intolerable delay. In the following, it is shown that to obtain $\{M_k\}_3$ for $k = 2, \ldots, 8, 9$, the calculation

$$\{M_k\}_3 = \frac{\{10kd\}_2 - \{5d\}_2}{10} \tag{43}$$

can be used instead. If (43) is to be an appropriate replacement for (42), it must satisfy the convergence condition (9) in the neighborhood of $\{M_k\}_3$. For simplicity, the investigation through (26) is limited to the positive quotient digits; however, inspecting the negative region gives identical results. Having used the truncation conditions (28) and (29), inequality $\{10w[j]\}_3 < \{M_k\}_3$, which corresponds to $q_{j+1} = k - 1$, can be expressed as $10w[j] - 10^{-3} < \{10w[j]\}_3 < \{M_k\}_3 < kd - (d/2) + 10^{-3}$ or equivalently

$$w[j+1] = 10w[j] - (k-1)d < -\frac{3d}{2} + 2 \times 10^{-3}. \tag{44}$$

To satisfy the convergence condition (3), (44) becomes $-(3d/2) + 2 \times 10^{-3} \leq d$ or correspondingly

$$d \geq 0.0008. \tag{45}$$

This is always correct since $d \geq (1/10)$. Repeating the inspection in the right neighborhood of $\{M_k\}_3$, where $q_{j+1} = k$ and $\{M_k\}_3 \leq \{10w[j]\}_3$, gives the same result. This means that (43) is a correct replacement for (42).

In the comparison multiples generator, all the division (multiplication) by ten operations are performed simply by one BCD wired right (left) shift, in zero time. Also, since the minuends and subtrahends are in the BCD format, as discussed in Section IV, the subtraction in (43) is carried out in zero time as well. However, since the final $\{M_k\}_3$ needs to be in the DSD format, only once during the initializing cycle, DSD to BCD conversion is required after subtraction (43). The conversion can be carried out using techniques, which are very similar to the ones developed for BSD to 2's complement conversion [35], [40]–[42]. These techniques have a delay of $\mathcal{O}(\log_2 W)$, where W is the width of the DSD operand in digits. The comparison multiple generator fulfills its tasks in the initializing cycle of DFP division (i.e., first cycle).

VII. DFP DIVISION EVALUATION

In this section, the latency and the area of the proposed DFP divider is estimated using a prelayout logical synthesis (Synopsys Design Compiler with Artisan 0.18-μm typical standard-cell library). It should be noted that the stages of unpacking the input operands and packing the quotient are not considered in the synthesis and, therefore, their delays do not participate in the estimation of the DFP division latency.

Investigation shows that the critical path delay of the design depicted in Fig. 2 is about 2.33 ns. This comprises the delays of 0.23 ns in the $q3$ register (including the setup and the hold times), 0.85 ns in the multiplexors, and 1.25 ns in the QDS*. Assuming the dividend and the divisor to be two decimal128 numbers with $p = 34$, then since the proposed DFP divider calculates the final quotient in $p + 3 = 37$ cycles, the division execution time is 86.2 ns.

The prelayout synthesis estimates an area of 48100 2-input NAND gates for the new comparison multiple based DFP divider. To the best of the authors' knowledge, this is only the second hardware DFP divider to be reported in the literature and the first to use the SRT algorithm. There are, therefore, few points of reference against, which to evaluate the divider; however, to help place it in the context it is interesting to note that one of the latest IEEE 754 compliant radix-4 FP dividers [43] occupies 4780 2-input NAND gates and has a latency of 72.6 ns (estimated using a 0.35-μm technology).

The first IEEE 754R compliant DFP divider published is due to Wang and Schulte [27]. It is based on the Newton-Raphson algorithm [14], which is different from the proposed DFP divider in nature. However, since both dividers accept the same inputs and generate the same output (in terms of the size and representation standard), it is reasonable to compare their latencies.

An estimated critical path delay of 0.69 ns is reported by Wang and Schulte [27]. The timing evaluation is obtained from a synthesis using Synopsys Design Compiler and LSI Logic 0.11-μm gflx-p standard cell library, under nominal operating conditions and a supply voltage of 1.2 V.

Wang and Schulte use a sequential fixed-point decimal multiplier [28] in the heart of the divider. It is able to produce between one and four digits per cycle. With a four-digit per cycle multiplier and for decimal128 dividend/divisor, Wang and Schulte report a DFP division latency of 113 cycles, which is equivalent to 77.97 ns. When they simulate the divider under more realistic conditions, i.e., an initial reciprocal approximation lookup table with a reasonable size and a one-digit per cycle multiplier, the Newton-Raphson-based DFP divider requires 246 cycles or 169.74 ns to produce the quotient.

The technologies used for timing evaluation in this paper and in [27] are different. Therefore, the absolute delays themselves could not be utilized for comparison. In order to make the delays comparable, the delays must be represented in FO4 [44]. Nedovic *et al.* [45] report 1 FO4 $\approx$ 45 ps measured by Fujitsu Laboratories for the 0.11-μm/1.2-V CMOS technology. For the 0.18-μm technology, 1 FO4 $\approx$ 65 ps. Having used these two numbers, the latency of the comparison multiple-based DFP divider is calculated as 1332 FO4, while the fastest and the slowest circuits reported by Wang and Schulte have the latencies of 1733 and 3772 FO4, respectively. Although these latencies include the delays of unpacking the input operands, handling the other rounding modes, and packing the result, the figures still show that the design proposed in this paper is faster than the latest DFP dividers reported in the literature.

VIII. Conclusion

Due to increasing demand for decimal arithmetic in financial and banking applications, developing circuits capable of performing arithmetic directly on decimal operands has become an important research topic. While addition, subtraction, and multiplication are much simpler to implement, implementing a divider accepting decimal dividends and divisors in the recently introduced IEEE 754R standard proposal is a great challenge.

A new DFP divider has been introduced. This uses DSD arithmetic, a new type of redundant decimal arithmetic. It also makes use of the comparison multiplies method for quotient digit selection. This method, which has been described both mathematically and as a VLSI architecture, calculates quotient digits in the sign-magnitude format, instead of searching for them in a lookup table. The QDS function receives a truncated PR and investigates the range to which the PR belongs. It performs the examination by comparing the truncated PR with the truncated multiples of the divisor produced once at the beginning of division. The result of the comparisons are delivered to a coder in order to produce the magnitude of the quotient digit. Meanwhile, another part of the QDS function, called the PR sign detector, calculates the polarity of the quotient digit by inspecting the sign of the truncated PR. In the comparison multiples-based QDS function, the PR sign detector operates off the critical path because the quotient digit sign and magnitude are calculated separately.

A logic synthesis with Artisan 0.18-μm typical standard-cell library indicates that the new DFP divider calculates the quotient's *coefficient* in 86.2 ns and occupies an implementation area equal to 48100 two-input NAND gates.

References

[1] M. D. Hill, N. P. Jouppi, and G. S. Sohi, *Readings in Computer Architecture*. San Mateo, CA: Morgan Kaufman, 2000.

[2] E. M. Schwarz, M. A. Check, C. L. K. Shum, T. Koehler, S. B. Swaney, J. D. MacDougall, and C. A. Krygowski, "The microarchitecture of the IBM eServer z900 processor," *IBM J. Res. Develop.*, vol. 46, no. 4/5, pp. 381–394, Jul./Sep. 2002.

[3] F. Y. Busaba, C. A. Krygowski, W. H. Li, E. M. Schwarz, and S. R. Carlough, "The IBM z900 decimal arithmetic unit," in *Proc. 35th Asilomar Conf. Signals, Syst. Comput. (ASILOMAR)*, 2001, pp. 1335–1339.

[4] A. Tsang and M. Olschanowsky, "A study of database 2 customer queries," Santa Teresa Lab., IBM, San Jose, CA, Tech. Rep. TR-03.413, 1991.

[5] D. M. Gay, "Correctly rounded binary-decimal and decimal-binary conversions," AT&T Bell Lab., Murray Hill, NJ, 1990, Numer. Anal. Manuscript 90-10.

[6] W. D. Clinger, "How to read floating point numbers accurately," in *Proc. Conf. Program. Lang. Des. Implementation*, 1990, pp. 92–101.

[7] European Commission Directorate General II, Brussels, Belgium, "The introduction of the euro and the rounding of currency amounts," (1999). [Online]. Available: http://europa.eu.int/comm/economy_finance/publications/euro_papers/20 01/eup22en.pdf

[8] I. S. Committee, "Some proposals for revising," ANSI/IEEE Std 754-1985, (2004). [Online]. Available: http://754r.ucbtest.org/

[9] S. Microsystems, "JSR-000013 Decimal arithmetic enhancement for the java programming language," (2001). [Online]. Available: http://jcp.org/aboutJava/communityprocess/review/jsr013/index.html

[10] P. S. Foundation, "Python 2.4," (2004). [Online]. Available: http://www.python.org/2.4/index.html

[11] M. F. Cowlishaw, "Decimal floating-point: Algorism for computers," in *Proc. 16th IEEE Symp. Comput. Arithmetic (ARITH-16)*, 2003, pp. 104–111.

[12] D. E. Atkins, "The theory and implementation of SRT division," Dept. Comput. Sci., Univ. Illinois at Urbana-Champaign, Urbana, IL, Tech. Rep. UIUCDCS-R-67-230, 1967.

[13] D. Harris, S. Oberman, and M. Horowitz, "SRT division architectures and implementations," in *Proc. 13th IEEE Symp. Comput. Arithmetic (ARITH-13)*, 1997, pp. 18–25.

[14] P. Soderquist and M. Leeser, "Area and performance tradeoffs in floating-point divide and square-root implementations," *ACM Comput. Surveys (CSUR)*, vol. 28, no. 3, pp. 518–564, 1996.

[15] S. F. Oberman and M. J. Flynn, "Design issues in division and other floating-point operations," *IEEE Trans. Comput.*, vol. 46, no. 2, pp. 154–161, Feb. 1997.

[16] H. Nikmehr, B. J. Phillips, and C. C. Lim, "A decimal carry-free adder," in *Proc. SPIE Conf. Smart Mater., Nano-, Micro-Smart Syst.*, 2004, pp. 786–797.

[17] M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Norwell, MA: Kluwer, 1994.

[18] H. Nikmehr, "Architectures for floating-point division," Ph.D. dissertation, Sch. Elect. Electron. Eng., Univ. Adelaide, Adelaide, Australia, 2005.

[19] S. F. Oberman, N. Quach, and M. J. Flynn, "The design and implementation of a high-performance floating-point divider," Dept. Elect. Eng. Comput. Sci., Stanford Univ., Stanford, CA, Tech. Rep. CSL-TR-94-599, 1994.

[20] S. F. Oberman and M. J. Flynn, "Division algorithms and implementations," *IEEE Trans. Comput.*, vol. 48, no. 8, pp. 833–854, Aug. 1997.

[21] *IEEE Standard for Binary Floating-Point Arithmetic*, 754-1985, 1985.

[22] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. London, U.K.: Oxford Univ. Press, 2000.

[23] M. D. Ercegovac and T. Lang, "On-the-fly rounding," *IEEE Trans. Comput.*, vol. 41, no. 12, pp. 1497–1503, Dec. 1992.

[24] H. Yabe, Y. Oshima, S. Ishikawa, T. Ohtsuki, and M. Fukuta, "Binary coded decimal number division apparatus," U.S. Patent 4,635,220, Jan. 6, 1987.

[25] A. Yamaoka, K. Wada, and K. Kuriyama, "Coded decimal non-restoring divider," U.S. Patent 4 692 891, Sep. 8, 1987.

[26] D. E. Ferguson, "Non-heuristic decimal divide method and apparatus," U.S. Patent 5 587 940, Dec. 24, 1996.

[27] L. K. Wang and M. J. Schulte, "Decimal floating-point division using Newton-Raphson iteration," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Arch., Processors (ASAP)*, 2004, pp. 84–95.

[28] M. A. Erle and M. J. Schulte, "Decimal multiplication via carry-save addition," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Arch., Processors (ASAP)*, 2003, pp. 348–359.

[29] B. Parhami, "Carry-free addition of recoded binary signed-digit numbers," *IEEE Trans. Comput.*, vol. C-37, no. 11, pp. 1470–1476, Nov. 1988.

[30] M. A. Thornton, "Signed binary addition circuitry with inherent even parity outputs," *IEEE Trans. Comput.*, vol. 46, no. 7, pp. 811–816, Jul. 1997.

[31] P. Kornerup, "Reviewing 4-to-2 adders for multi-operand addition," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Arch. Processors (ASAP)*, 2002, pp. 218–229.

[32] M. Cowlishaw, "Decimal arithmetic encoding strawman 4D," (2003). [Online]. Available: http://www2.hursley.ibm.com/decimal/decbits.pdf

[33] ——, "Densely packed decimal encoding," *Proc. IEE —Comput. Dig. Techn.*, vol. 142, no. 3, pp. 102–104, May 2002.

[34] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. San Mateo, CA: Morgan Kaufman, 2004.

[35] K. K. Parhi, "Fast low-energy VLSI binary addition," in *Proc. IEEE Conf. Comput. Des.*, 1997, pp. 676–684.

[36] G. M. Blair, "The equivalence of Twos-complement addition and the conversion of redundant-binary to Twos-complement numbers," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 45, no. 6, pp. 669–671, Jul. 1998.

[37] R. Zimmermann, "Binary adder architectures for cell-based VLSI and their synthesis," Ph.D. dissertation, Swiss Federal Inst. Technol. Zurich, Zurich, Switzerland, 1998.

[38] J. S. Chiang, H. D. Chung, and M. S. Tsai, "Carry-free radix-2 subtractive division algorithm and implementation of the divider," *Tamkang J. Sci. Eng.*, vol. 3, no. 4, pp. 249–255, 2000.

[39] J. D. Bruguera and T. Lang, "Multilevel reverse-carry addition: Single and dual adders," *J. VLSI Signal Process.—Syst. Signal, Image, Video Technol.*, vol. 33, no. 1–2, pp. 55–74, Jan.–Feb. 2003.

[40] M. Uya, K. Kaneko, and J. Yasui, "A CMOS floating-point multiplier," *IEEE J. Solid State Circuits*, vol. SC-19, no. 5, pp. 697–702, Oct. 1984.

[41] H. R. Srinivas and K. K. Parhi, "A fast VLSI adder architecture," *IEEE J. Solid-State Circuits*, vol. 27, no. 5, pp. 761–768, May 1992.

[42] A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer, and P. G. A. Jespers, "A new carry-free division algorithm and its application to a single-chip 1024-b RSA processor," *IEEE J. Solid-State Circuits*, vol. 25, no. 6, pp. 748–755, Jun. 1990.

[43] E. Antelo, T. Lang, P. Montuschi, and A. Nannarelli, "Fast radix-4 retimed division with selection by comparisons," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Arch., Processors (ASAP)*, 2002, pp. 185–196.

[44] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.

[45] N. Nedovic, W. W. Walker, V. G. Oklobdzija, and M. Aleksic, "A low power symmetrically pulsed dual edge-triggered flip-flop," in *Proc. 28th Eur. Solid-State Circuits Conf.*, 2002, pp. 399–402.

[46] J. Cocke and D. W. Sweeney, "High speed arithmetic in a parallel device IBM, Corporation, San Jose, CA, 1957.

[47] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electron. Comput.*, vol. EC-7, no. 3, pp. 88–92, Sep. 1958.

[48] K. D. Tocher, "Techniques of multiplication and division for automatic binary computers," *Quarterly J. Mech. Appl. Math.*, vol. 11, pp. 364–384, 1958.

Hooman Nikmehr received the B.Sc. degree in electronic engineering and M.Sc. degree in computer architecture engineering from the University of Tehran, Tehran, Iran, in 1992 and 1997, respectively, and the Ph.D. degree in computer architectures and systems from the University of Adelaide, Adelaide, Australia, in 2005.

He has been a Lecturer with the Department of Computer Engineering, Bu-Ali Sina University, Hamedan, Iran, since 1997. His current research interests include VLSI, digital arithmetic, computer architecture, and low-power design.

Braden Phillips (M'93) received the Ph.D. degree from the University of Adelaide, Adelaide, Australia, in 2001. His thesis was entitled "An Optimised Implementation of Public Key Cryptography for Smart Card Processors."

He is currently a Lecturer in the School of Electrical and Electronic Engineering, University of Adelaide. He worked as a process control engineer and was a founding partner in Current Dynamics, an electronic hardware design venture. In September 2000, he took up a lecturing position at Cardiff University, Cardiff, U. K., in South Wales, a post he held for two years before returning to Adelaide. His research interests include digital arithmetic, digital microelectronics, computer architecture, real-time systems, and information security.

Cheng-Chew Lim (M'82–SM'02) received the B.Sc. degree (with honors) in electronic and electrical engineering and the Ph.D. degree from Loughborough University, Leicestershire, U.K., in 1977 and 1981, respectively.

He is currently an Associate Professor and Reader in the School of Electrical and Electronic Engineering, University of Adelaide, Adelaide, Australia. His research interests include VLSI architectures for matrix computation, multi-channel digital receivers, machine learning, and control system.