Published in IET Computers & Digital Techniques
Received on 17th March 2010
Revised on 21st July 2010
doi: 10.1049/iet-cdt.2010.0026


ISSN 1751-8601

# Improving the speed of decimal division

*A. Kaivani*[1] *A. Hosseiny*[1] *G. Jaberipur*[1,2]

[1]*Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran*
[2]*School of Computer Science, Institute for Research in Fundamental Sciences (IPM), P.O. Box: 19395-5746, Tehran, Iran*
*E-mail: a_kaivani@sbu.ac.ir*

**Abstract:** The authors study previous major contributions to digit recurrence decimal division hardware and focus on techniques for improving the performance of quotient digit selection (QDS) as the most complex part. In particular, Design D1 uses the digit set [−5, 5] for quotient digits. Another design (D2) uses mixed binary/decimal carry-save manipulation of the few most significant digits of partial remainders. Motivated by successful combined arithmetic algorithms such as hybrid adders, the authors offer a decimal division scheme that takes advantage of the best design options of D1 and D2 with due modifications that significantly enhance the division speed. In particular, they configure the architectures of QDS and partial remainder computation paths in favour of reduced balanced latencies of both. Furthermore, they remove the rounding cycle by cost-free auto-rounding, which is an exclusive advantage of the digit set [−5, 5]. The authors of D1 and D2 have used logical effort (LE) and circuit synthesis to evaluate their dividers, respectively. Therefore for a fair comparison, the authors evaluate the proposed design (D3) with both methods. The LE-based D3/D1 comparison shows 21% more speed at the cost of 6% more area, whereas the synthesis-based D3/D2 comparison results in 46% less latency and 23% less area.

## 1 Introduction

Division is the most difficult of the four basic arithmetic operations and even more difficult in decimal computer arithmetic. The digit recurrence class of division hardware algorithms generally mimics the conventional paper and pencil division scheme as taught in elementary school [1]. In these iterative algorithms, one quotient digit is determined per iteration such that the division latency is linearly dependent on the number of quotient digits.

There are, however, multiplicative division algorithms that progressively produce an approximation of the quotient, where the number of iterations is logarithmically proportional to the quotient word size [1]. During the past few decades, several division schemes from both digit recurrence and multiplicative classes have been implemented in hardware (e.g. [2–8]).

With the revived popularity of decimal arithmetic hardware units in digital processors [9], a series of innovations is ongoing in the design of decimal division schemes for implementation on binary processors (e.g. [5] for multiplicative class and [6–8] that focus on digit recurrence division). The quotient digit selection (QDS) for decimal division, naturally more complex than that of binary, is the main source of difference among several digit recurrence approaches. Two techniques have been used in the just-cited recent contributions for decimal QDS simplification. One is to use a signed digit set (e.g. [−5, 5] in [8]) for quotient representation. This leads to fewer comparisons with divisor multiples. The other is to introduce more redundancy in the digit set. This allows extra simplifying imprecision (e.g. [−7, 7] in [7] and [−9, 9] in [6]). The decimal division unit of the IBM Power6 processor [10] uses an area-improved QDS technique [11], where the next quotient digit is simply determined by rounding the partial remainder. However, this fast single cycle 16-digit QDS is achieved at the cost of 12 cycles required for prescaling the divisor to approach 1.0.

One of the design strategies that occasionally leads to performance improvement is to take advantage of the best components of two or more successful designs. Possible modification of selected components, addition of new components and finally assembling a design can lead to enhanced performance with respect to one or more of performance measures. Examples include, hybrid adders [12], residue number system multiplication [13], the computation of square root [14] and decimal multiplication [15], to name a few. In this paper, following the methodology of the afore-cited works that use a hybrid or combined design approach, we present a digit recurrence decimal division scheme that is based on the following three sets of design properties. Sets I and II are due to [7] and [8], respectively. Set III summarises the proposed new techniques and improvements.

I. Selected design properties of [7] (see Section 4.3 for details):

- Overlapped implementation of QDS and partial remainder computation (PRC);
- Low-latency manipulation of partial remainders using redundant adders;
- QDS simplification via mixed binary/decimal representation of the few most significant digits of partial remainders that imposes hardware redundancy.

II. Selected design properties of [8] (see Section 4.4 for details):

- Overlapped implementation of QDS and PRC;
- Quotient representation with the minimally redundant signed digit set $[-5, 5]$;
- Direct generation of comparison and divisor multiples.

III. Main design properties of this work:

- *Balanced QDS and PRC:* We combine and improve the above techniques in order to reduce and balance latencies for QDS and PRC (Section 3.1).
- *Additional QDS simplification:* The last part of QDS, which selects the appropriate multiple of the divisor and the quotient digit, is considerably faster via simply generated enable signals (see Section 3.1.2 for details).
- *Partitioned binary/decimal PRC:* This design option obviates the need for PRC hardware redundancy that is the case in [7]. In addition, a special technique is used for multiplication by 10 of binary part of PRC (see Section 3.1.1).
- *Initialisation:* The binary QDS and partitioned binary/decimal PRC have affected our initialisation method, where we use no look-up tables (see Section 3.2).
- *Auto-rounding:* The digit set $[-5, 5]$, used in the proposed scheme, allows for automatic balanced-error rounding (see Section 3.3).

The rest of this paper is organised as follows. We present a background on digit-recurrence division in Section 2 and discuss the proposed improvements on the decimal division hardware architecture in Section 3. Section 4 provides comparison between the proposed design and those of the previous works, and finally Section 5 draws our conclusions. Table 1 describes the symbols and abbreviations that are used throughout the paper, where the number representation system is also provided if applicable.

## 2 Background

Equation (1) provides an abstract description of digit recurrence (or subtractive) division, where the dividend $\Gamma$ and divisor $\Delta$ are the input operands and the division algorithm is to compute the quotient $Q$ and remainder $R$

$$\Gamma = Q \cdot \Delta + R \tag{1}$$

The hardware implementation of subtractive division requires a normalised dividend and divisor [16]. This is because of practical reasons that arise in floating point arithmetic. For example, we assume $0.1 \leq \Gamma, \Delta < 1$ for decimal division. Equation (1) is iteratively implemented via a digit recurrence as outlined in (2). A description of each of the symbols used is found in Table 1 below. The quotient digit $q_{i+1}$ is computed by the QDS unit, and the rationale for the initial conditions will be explained later (Section 3.2).

$$\begin{aligned} \Omega[i+1] &= 10\Omega[i] - q_{i+1}\,\Delta; \quad \text{for } i \geq -1, \\ \Omega[0] &= 10\Omega[-1] = \Gamma/100, \quad q_0 = 0 \end{aligned} \tag{2}$$

### 2.1 QDS

It is well known [16] that QDS is simplified if quotient $Q$ is represented as a redundant signed digit number with digit set $[-\alpha, \alpha]$, where the redundancy factor is defined as $\rho = \alpha/9$ for radix-10 division [7]. Furthermore, the conditions required for the convergence of recurrence (2) is

**Table 1** Symbols and abbreviations

| Symbol/abbreviation | Description | Representation |
|---|---|---|
| BCD | binary coded decimal | N/A |
| BCD-FA | BCD full adder | N/A |
| DSD | decimal signed digit | N/A |
| 2CL | two's complement | N/A |
| CS | carry save | N/A |
| 2CLCS | two's complement CS | N/A |
| QDS | Quotient digit selection | N/A |
| PRC | partial remainder computation | N/A |
| LE | logical effort | N/A |
| AutoR | auto-rounding | N/A |
| IEEER | IEEE rounding | N/A |
| $\Gamma$ | dividend | BCD |
| $\Delta$ | divisor | BCD |
| $d$ | binary part of divisor | 2CL |
| $D$ | decimal part of divisor | BCD |
| $\Omega[i]$ | $i$th partial remainder | CS |
| $w[i]$ | binary part of $i$th partial remainder | 2CLCS |
| $W[i]$ | decimal part of $i$th partial remainder | BCD-CS |
| $\Omega_k[i]$ | $10\Omega[i-1] - k\Delta$ | CS |
| $q_i$ | $i$th quotient digit | minimally redundant decimal |
| $M_K\left(m_k = \frac{M_k}{10}\right)$ | $k$th (reduced) comparison multiple | 2CL |
| $\rho$ | redundancy factor | N/A |
| $n$ | number of decimal digits | N/A |
| $V'$ | approximation of $V$ | N/A |
| posibit ● | positive bit {0, 1} | single bit |
| negabit ○ | negative bit {−1, 0} | single bit |



*IET Comput. Digit. Tech.*, 2011, Vol. 5, Iss. 5, pp. 393–404
doi: 10.1049/iet-cdt.2010.0026

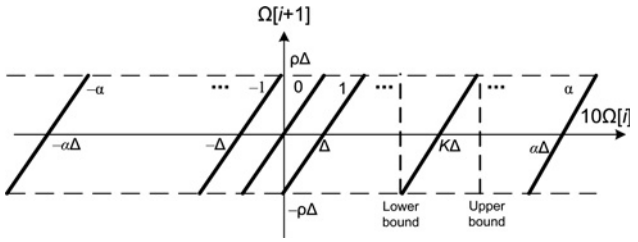


**Fig. 1** *Overlapped regions of (3)*

defined, as in (3), and illustrated in Fig. 1 [16].

$$-\rho\Delta \le \Omega[i+1] = 10\Omega[i] - q_{i+1}\Delta \le \rho\Delta \tag{3}$$

It is common to select $q_{i+1}$ by checking the 10-fold of partial remainder $10\Omega[i]$ against some comparison multiples $M_k$ for $k \in [-\alpha, \alpha]$, as in (4). The largest $k$, for which $\Omega_k[i+1] > 0$ determines the values of $\Omega[i+1]$ and $q_{i+1}$, as described by (5) [16].

$$\Omega_k[i+1] = (10\Omega[i]) - M_k \tag{4}$$

$$\Omega[i+1] = \Omega_k[i+1] \quad \text{and} \quad q_{i+1} = k, \quad \text{if } \Omega_k[i+1] \ge 0 \quad \text{and} \quad \Omega_{k+1}[i+1] < 0 \tag{5}$$

To make QDS less costly it is desired to perform all computations with $t$-fractional-digits (i.e. with fixed-point numbers with $t$ digits to the right of the radix point). Let $V'$ denote $V$ truncated into $t$-fractional-digits. For example, we can use $\Omega'_k[i+1]$ as a truncated approximation of $\Omega_k[i+1]$ such that $\Omega'_k[i+1] > \Omega_k[i+1] - 10^{-t}$. Consequently, (6) and (7) can serve for QDS via truncated variables. However, it has been shown that such approximation may lead to an incorrect value for $q_{i+1}$ [16]. One way to remedy this problem is to select the comparison multiples such that they fall within the overlapped regions (see Fig. 1) and thus be bound as in (8) [17]. This approach forces the division algorithm to converge in spite of the approximation error.

$$\Omega'_k[i+1] = (10\Omega[i])' - M_k \tag{6}$$

$$q_{i+1} = k, \quad \text{if } \Omega'_k[i+1] \ge 0 \quad \text{and} \quad \Omega'_{k+1}[i+1] < 0 \tag{7}$$

$$(k-\rho)\Delta \le M_k \le (k-1+\rho)\Delta \tag{8}$$

Following QDS (i.e. determination of $q_{i+1}$), the PRC via full-word subtraction, based on (2), leads to the new partial remainder, $\Omega[i+1]$. Fig. 2 provides an abstract illustration of non-overlapping QDS and PRC. However, the designs by [6] and [7] overlap the computations of QDS and PRC.

It is naturally desirable to keep $t$ (i.e. the number of fractional digits in the truncated partial remainders) as low as possible. Equation (9) describes $t$ in terms of $\alpha$ for radix 10, where the derivation details, based on [7] and [17], are offered in Appendix 1

$$t = 1 + \left\lceil \log_{10}\left(\frac{36}{2\alpha - 9}\right)\right\rceil \tag{9}$$

It turns out that minimum $t$ for $\alpha \in \{5, 6\}$ is equal to 3; it is 2 for $\alpha \in [7, 22]$ and 1 for $\alpha \ge 23$. Note that for larger $\alpha$ the advantage of faster QDS, because of smaller $t$, should be carefully evaluated against the larger area and delay for generation of multiples.

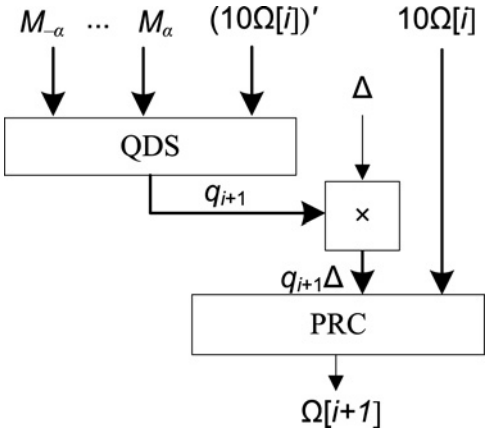


**Fig. 2** *QDS (6) and (7) and PRC (2)*

### 2.2 Decimal division with quotient digits in [−9, 9]

The decimal digit-recurrence division algorithm of [6] uses the redundant decimal signed digit (DSD) set [−9, 9] to represent the digits of quotient and partial remainders. Given that $\alpha = 9$, the number $t$, of (9), is equal to 2. Each DSD is encoded via two equally weighted positive and negative BCD digits. The comparison multiples are stored in a look-up-table. A special redundant QDS adder receives the partial remainder and a multiple of the devisor, as DSD operands, and the comparison multiple in standard BCD format. Finally, standard on-the-fly conversion [18] is used for converting the quotient into non-redundant format.

### 2.3 Decimal division with quotient digits in [−7, 7]

Lang and Nannarelli [7] presented a digit recurrence decimal division algorithm, where the quotient digits are represented with redundant decimal digit set [−7, 7] and partial remainders are maintained in decimal carry-save encoding [7]. Each quotient digit, $q_i$, is decomposed into a BSD digit, $q_{i_H} \in [-1, 1]$, and a minimally redundant radix-4 digit, $q_{i_L} \in [-2, 2]$, such that $q_i = 5 \times q_{i_H} + q_{i_L}$. This clever observation has reduced the required multiples of the devisor to only $\pm\Delta$, $\pm 2\Delta$ and $\pm 5\Delta$, and these can be computed via fast carry-free logic [19]. One drawback is that the PRC requires two subtract operations (e.g. for $q_i = 7$, $5\Delta$ and $2\Delta$ should be subtracted from partial remainder). However, the cost of this is reduced using BCD carry-save manipulation. The three most significant digits, including two fractional ones (by (9), $t = 2$ for $\alpha = 7$) that participate in QDS are maintained in binary carry-save format for faster QDS manipulation. Look-up table retrieval, of comparison multiples, and on the fly conversion and normalisation of quotient digits are used.

### 2.4 Decimal division with quotient digits in [−5, 5]

Vazquez *et al.* [8] have presented a decimal divider, where the minimally redundant DSD set [−5, 5] is used to represent quotient digits. Aiming at an area-optimised design, they keep the partial remainders in non-redundant format with 5-2-1-1 encoding of decimal digits. For the same reason (i.e. area optimisation), the comparison multiples (i.e. $M = \{\pm 0.5\Delta, \pm 1.5\Delta, \pm 2.5\Delta, \pm 3.5\Delta, \pm 4.5\Delta\}$), contrary to the previous works, are generated in the initialisation cycle by means of combinational circuits.

## 3 Proposed new decimal divider

The proposed digit-recurrence decimal divider mainly follows that of [7], but uses the minimally redundant decimal digit set $[-5, 5]$, as in [8]. Therefore we take advantage of overlapped QDS/PRC, mixed binary/decimal PRC, carry-save partial remainder encoding, on the fly quotient conversion and normalisation and the smaller digit set $[-5, 5]$ that leads to less costly QDS and generation of comparison multiples. Furthermore, we balance the overlapping between the PRC and QDS, with the effect of reducing the QDS latency and increasing that of PRC. Finally, the proposed so-called auto-rounding technique obviates the need for the extra rounding cycle(s). We describe the proposed decimal division scheme in three parts, namely:

- *Initialisation:* Generation of comparison and divisor multiples and other preparations for the recurrence part;
- *Recurrence:* QDS and PRC;
- *Clean up:* Normalisation, conversion, correction and rounding of the quotient.

We start with description of the most important part, the recurrence implementation. The presented material supports the other two parts (i.e. initialisation and clean up) to be discussed in the subsequent Sections 3.2 and 3.3.

### 3.1 Recurrence

The recurrence equation (i.e. (2) in the prologue of Section 2) suggests that $q_{i+1}$ should be selected first, followed by computation of the next partial remainder (Fig. 2). The QDS is based on (8), where the approximated values are represented with only four decimal digits. That is one digit before the radix point and three fractional digits due to $t = 3$, where the latter is obtained from (9) for $\alpha = 5$. As in [7], we maintain the approximate partial remainders, divisor and comparison multiples in binary format. However, the PRC must result in full word partial remainders; hence, the mixed binary/decimal format, where only the binary part is used in QDS. This keeps the decimal part of PRC off the critical path.

To speed up each recurrence step, one can overlap the PRC with QDS for the next quotient digit, as in Fig. 3, which describes (8) (for $q_{i+2}$) and (10), where the primed expressions represent truncated approximations as before.

$$\begin{aligned}\Omega'_k[i+2] &= (10\Omega[i+1])' - M_k = (10(10\Omega[i] - q_{i+1}\Delta))' - M_k \\ &= (100\Omega[i])' - (10q_{i+1}\Delta)' - M_k\end{aligned} \tag{10}$$

Note that the first term in (10) (i.e. $(100\Omega[i])'$) must include five decimal digits; two digits before the radix point and three fractional digits because of $t = 3$. The multiplication by 100 in binary representation imposes additional area and delay. In order to avoid the latter, as in [7], we use reduced comparison multiples (i.e. $m_k = M_k/10$) and compute $q_{i+2}$ as in (11). The reduced multiples ($m_k$) lead to $t = 4$; hence 5 decimal digits are again required. The work of [7] retrieves 14 reduced comparison multiples from look-up table. However, we compute the ten required reduced comparison multiples via combinational logic to be described in Section 3.2.3. As such, we save area without any effective latency overhead.

$$\begin{aligned}&q_{i+2} = k, \\ &\quad \text{if } \Omega'_k[i+2] = (10\Omega[i])' - (q_{i+1}\Delta)' - \frac{M_k}{10} \geq 0 \\ &\quad \text{and} \quad \Omega'_{k+1}[i+2] < 0\end{aligned} \tag{11}$$

In order to speed up the computation of $\Omega'_k[i+2]$ (for $-\alpha < k \leq \alpha$) we, as in [7], maintain it in 2CLCS representation. To achieve this, instead of area intensive and slow look-up table retrieval, we precompute the divisor multiples and generate the reduced comparison multiples in two's complement. Therefore, the main computation in (11) can be performed via a network of (4:2) compressors with four input binary operands (i.e. two because of the 2CLCS representation of $(10\Omega[i])'$ and two for the binary representations of $-(q_{i+1}\Delta)'$ and $-M_k/10$). Quotient digit $q_{i+2}$ can now be selected via sign detection of the $\Omega'_k[i+2]$ values, for $-\alpha < k \leq \alpha$, as is prescribed by (11); details to come later in Section 3.1.2.

Fig. 4 depicts the redundant binary/decimal partitioning of the partial remainders for 16-digit operands. The five most significant digits (i.e. the binary part $(10\Omega[i])'$) are represented in 2CLCS as $10w[i]$. The rest of the digits are kept in carry-save BCD as $10W[i]$. Only the bits of $10w[i]$ (i.e. the $b$ and $b'$ bits) take part in QDS.

Given that the number of truncated fractional digits in $(10\Omega[i])'$ was determined to be $t = 4$, the number of binary positions in $10w[i]$ is figured out as follows:

Recalling the convergence condition, described by (3), given the normalised divisor (i.e. $\Delta < 1$) and the digit set $[-5, 5]$, we obtain at the actual convergence condition shown by (12), where $\rho = \alpha/(r-1) = 5/9$. Therefore the range of the binary part (i.e. $10w[i]$) is determined by (13). This range of values, given that $2^{15} < 55555 < 2^{16}$, requires 17 bits in binary representation and, at most, 17 bits in 2CLCS (Fig. 4).

$$|10\Omega[i]| \leq 10\rho\Delta = \frac{50}{9}\Delta \leq \frac{50}{9} = 5.5555 \tag{12}$$

$$-5.5555 \leq 10w[i] \leq 5.5555 \tag{13}$$

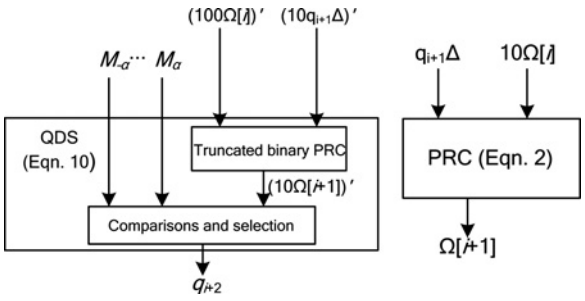


**Fig. 3** *Fully overlapped QDS and PRC*

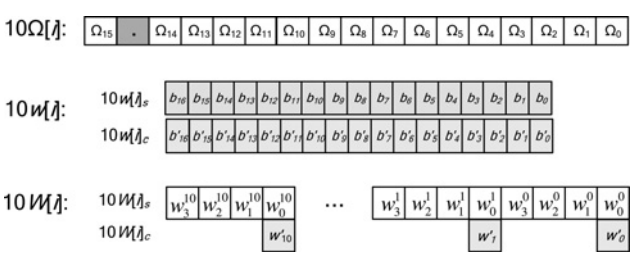


**Fig. 4** *Redundant binary/decimal partitioned representation of partial remainders*




doi: 10.1049/iet-cdt.2010.0026

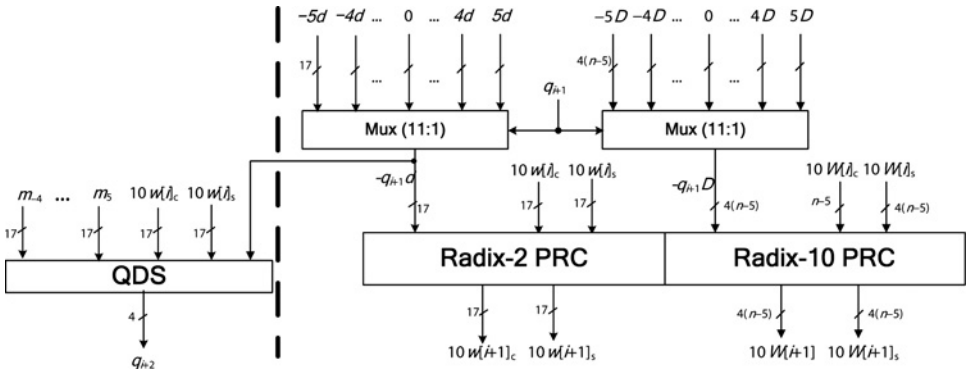


**Fig. 5** *Binary and decimal QDS and PRC*

Fig. 5 depicts the abstract architecture of binary and decimal PRC and the QDS, where symbols $d$ and $w$ ($D$ and $W$), denoting binary (decimal) values, were defined in Table 1.

*3.1.1 PRC:* The inputs and outputs of the binary and decimal carry-save adders that compute the two portions of the partial remainders are shown in the right part of Fig. 5. The decimal (binary) CSA implements (14) [(15)], which is a reproduction of (2) for decimal (binary) part and a final multiplication by 10.

$$W[i+1] = 10W[i] - q_{i+1}D \tag{14}$$

$$w[i+1] = 10w[i] - q_{i+1}d \tag{15}$$

3.1.1.1 The decimal PRC: The required decimal PRC of Fig. 5 can be implemented using a decimal CSA composed of $(n-5)$ BCD full adders (BCD-FA), as shown in Fig. 8 (recall that [7] uses $n$ BCD-FAs). However, given the final $\times 10$ operation after performing (14), the operands of the BCD-FA in position $(n-6)$ are kept intact and, along with another decimal carry coming from the next lower decimal position, are fed into the binary PRC part (see details in Section 3.1.1.2). A BCD-FA receives two BCD digits and a decimal carry (e.g. a digit of $10W[i]_s$, a digit of $-q_{i+1}D$ and a carry from $10W[i]_c$ in Fig. 5) and produces one BCD sum digit and a decimal carry to be stored along with the next higher sum digit. We use the BCD-FA of [20].

3.1.1.2 The binary PRC: Fig. 6 depicts the operation details of the binary PRC of Fig. 5 in extended dot notation. The circuitry is shown within the left dashed box in Fig. 8.

Part I of Fig. 6 illustrates the three 17-bits input operands, where a black (white) dot indicates a posibit (negabit). In preparation for the next recurrence, the result in Part II is multiplied by 10 (Part III) as $10w[i+1] = 8w[i+1] + 2w[i+1]$, where the one-bit- and the three-bit-shifted sum and carry parts of $w[i+1]$ serve as inputs to a network of universal (4:2) compressors that accept any four-deep mix of posibits and inversely encoded negabits [21]. To ease the migration of negabits to the most significant position of the result we sign-extend the top operand by one bit. The leftmost cell (oval) is actually a full adder that receives a negabit from the right (4:2) compressor. Owing to the required $\times 10$ operation (after (14)), the result in Part IV is augmented by $10W[i]_S^{n-6} + 10W[i]_c^{n-6}$ and the most significant digit of $-q_{i+1}D$ from the decimal PRC and a shifted carry out of position $(n-7)$.

A 16-bit CSA takes care of the 37-bit collection in positions 0–15 of Part IV that leads to the result in the same positions of Part V and a carry to position 16. The leftmost 10-bit collection remains intact. This result in Part V is too wide to be fed in as the input for the next

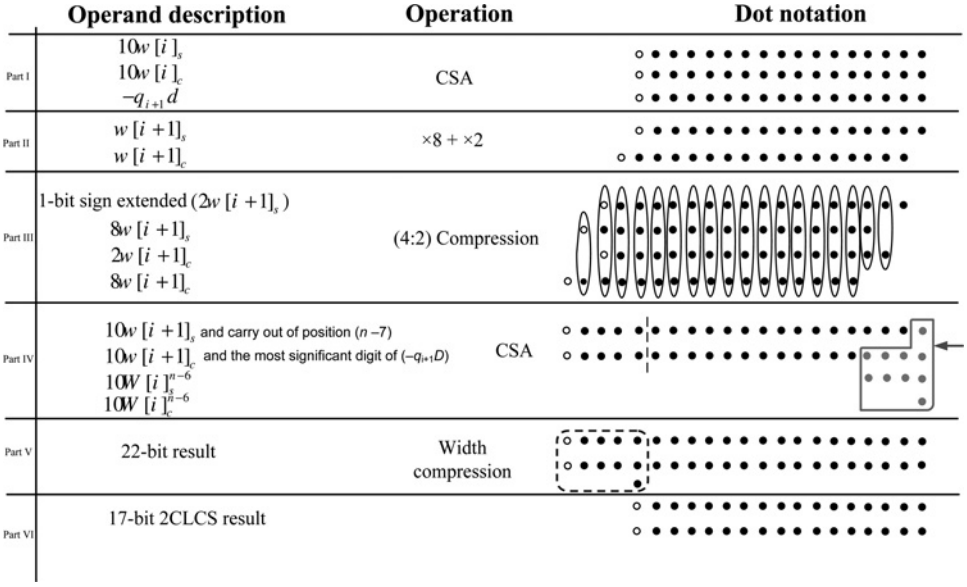


**Fig. 6** *Five steps of binary PRC*

recurrence that accepts 17-bit 2CLCS numbers as binary partial remainder. Fortunately, however, it turns out that the total arithmetic value of the bits in positions 16 and beyond falls in $[-2, 0]$. Therefore they can be represented by two equally weighted negabits in position 16. The reason is given below.

Let the collective value of the bits in Part V of Fig. 6 (equal to that of $10w[i+1]$), disregarding the radix point, be denoted by $2^{16}e + E$, where $e$ and $E$ relate to the bits within the dashed box and the rest of least significant bits, respectively. Then, given the bounds in (13), the following holds. Replacing for $E$ in the left inequality of equation set 16 leads to $-2 \leq e \leq 0$.

$$-2^{16} < -55555 \leq 2^{16}e + E \leq 55555 < 2^{16},$$
$$0 \leq E \leq 2^{17} - 2 \tag{16}$$

The logical equations that describe the required circuitry for width compression are given in Appendix 2.

*3.1.2 QDS:* Recalling (10) for computation of $q_{i+2}$ and the decomposition of partial remainders and divisor multiples into binary and decimal parts, as in Fig. 5, operation of the QDS box of that figure can be described by (17).

$$q_{i+2} = k, \quad \text{if } \frac{M_k}{10} \leq (10w[i] - q_{i+1}d) < \frac{M_{k+1}}{10} \tag{17}$$

To implement (17), we determine the sign of $w_k[i]$ defined by (18) for $-5 < k \leq 5$.

$$w_k[i] = 10w[i]_c + 10w[i]_s - \frac{M_k}{10} - q_{i+1}d \tag{18}$$

The latter can be efficiently computed by a network of (4; 2) compressors. To determine the value of $k$, for which the sign of $w_k[i]$ changes, we use a sign detection parallel prefix network to determine the sign of all $w_k[i]$. Then we use these sign signals to produce enable signals (Fig. 7) for selecting the divisor multiples. Furthermore, we use a special 10-to-4 priority encoder that receives 10 sign signals (for $-5 < k \leq 5$) and provides a four-bit two's complement number $q_{i+2} \in [-5, 5]$. Note that, as will be clarified below, this encoder is off the critical delay path.

To decide on the critical delay path of Fig. 8, we use a rough gate level delay evaluation of the PRC and QDS, where the delay of simple AND/OR gates with at most three inputs is denoted by ΔG, that of a NOT is 0.5 ΔG and 1.5 ΔG is considered for an XOR gate.

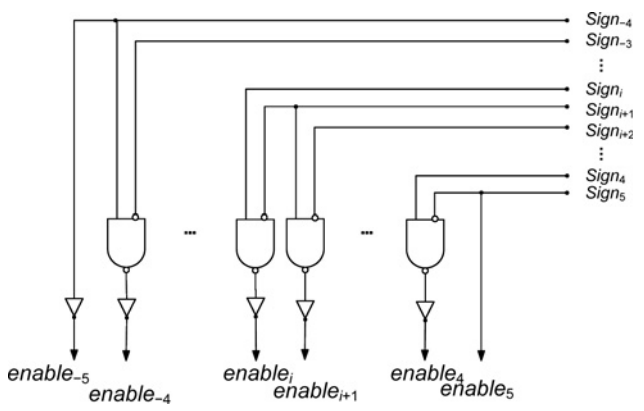


**Fig. 7** *Enable signal generator (ESG)*

The overall delay of the PRC is 19 ΔG that is composed of the following delay components related to the parts of Fig. 6:

- One full adder for the CSA of Part I (3 ΔG);
- One NOT for part II (0.5 ΔG);
- Three XORs for the (4:2) compressors of Part III (4.5 ΔG);
- One full adder for the CSA of Part IV (3 ΔG);
- The effective delay of special logic for Part V (8 ΔG).

The overall QDS delay is 21 ΔG that is composed of the following delay components:

- Three XORs for the (4:2) compressors of (18) (4.5 ΔG);
- The delay of sign detection parallel prefix network (11 ΔG);
- The delay of the ESG (1.5 ΔG);
- The delay of the selector of the divisor multiple (4 ΔG).

Therefore we have a reduced relatively balanced-delay PRC and QDS, where the dashed lines of Fig. 8 show the two critical delay paths. We will determine the actual critical delay path of the whole circuit using logical effort (LE) analysis and synthesis tool in Section 4.1.

### 3.2 Initialisation

In this section, we describe the following initialising steps, where we assume conventional BCD encoding for the dividend and divisor:

- Decomposition of the first partial remainder $\Omega[0] = 10\Omega[-1] = \Gamma/100$ (see (2)) to binary and decimal parts;
- Precomputation of the divisor multiples $-5\Delta, -4\Delta, \ldots, 4\Delta, 5\Delta$;
- Decomposition of the divisor multiples to binary $(-5d, \ldots, 5d)$ and decimal multiples $(-5D, \ldots, 5D)$ such that $\Delta = 10^{n-5}d + D$ disregarding the decimal point;
- Generation of comparison multiples $m_{-4}, \ldots, m_5$.

Implementation of the above steps turns out to require more design effort than that of the similar tasks in [7, 8]. The reason is that the former simply uses an area intensive and slow look up table to retrieve the required multiples and the latter precomputes only the decimal multiples. In spite of additional complexity, we have only one cycle exclusively dedicated to initialisation.

*3.2.1 Generating the first partial remainder:* Recall that in the principal QDS equation (i.e. (17)) the quotient digit $q_{i+2}$ is expressed in terms of $w[i]$. Therefore the index of initial values has to be $i = -1$ to allow for $q_1$ to be the first computed quotient digit. Also recalling (2), we need to initialise $10\Omega[-1]$ as the first partial remainder to be fed to the PRC as the binary $10w[-1]$ and decimal $10W[-1]$; so is fed the initial quotient digit $q_0 = 0$. The first round of PRC will lead to computation of $\Omega[0] = 10\Omega[-1] - q_0\Delta$, according to (2). The convergence (3), $i = -1$ and $\rho = 5/9$, should hold for all values of Δ, including $\Delta = 0.1$. This leads to $\Omega[0] < 5/90 < 0.056$. Therefore given that $\Gamma < 1$, we can let $\Omega[0] = \Gamma/100 < 0.01 < 0.056$. The latter choice allows for easy initialisation via two right decimal shifts of Γ. However, given that $q_0 = 0$, an extra recurrence step and two final simple decimal left shifts are required. Moreover, it is interesting to note that $q_1 \in \{0, 1\}$, which can be easily




doi: 10.1049/iet-cdt.2010.0026

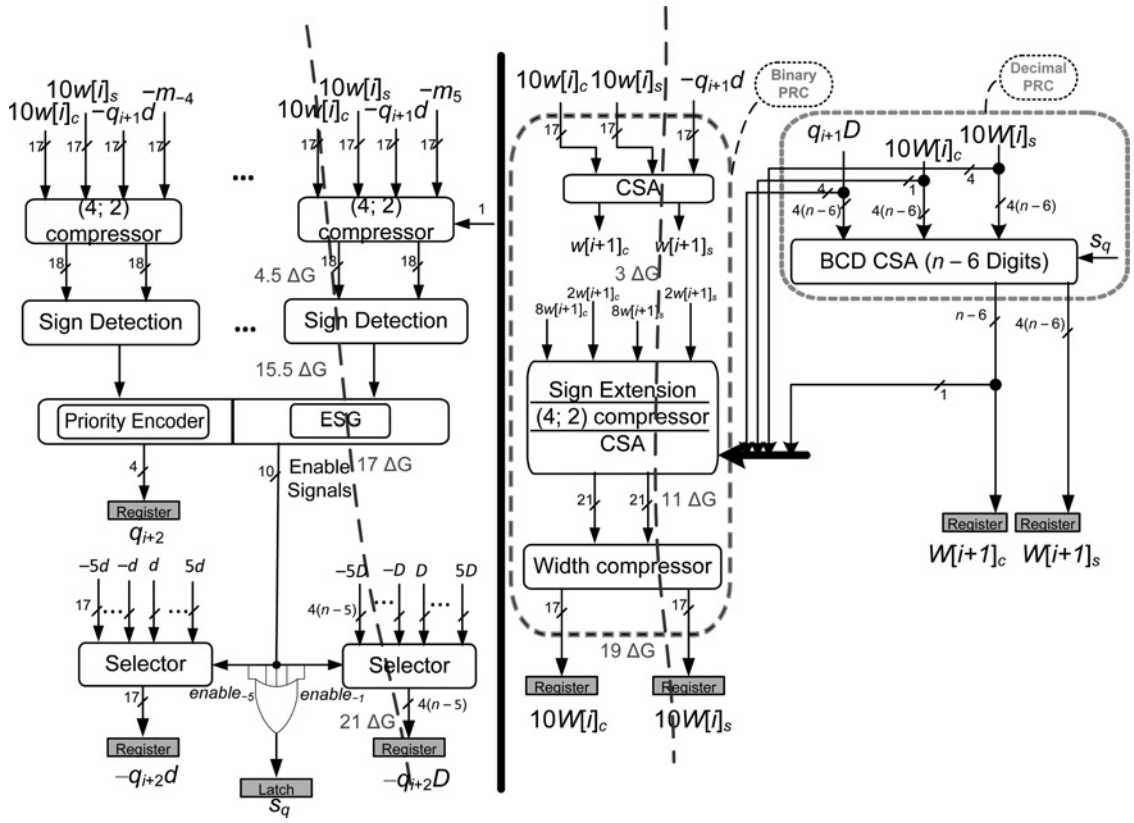


**Fig. 8** *Reduced latency-balanced QDS and PRC of the proposed decimal divider*

verified by consulting the diagram of Fig. 1 and applying the above initialisation on (2) (see also Sections 3.2.2 and 3.2.3).

Recalling that $\Gamma = 0.\gamma_1\gamma_2\ldots$ and $\Omega[0] = 10\Omega[-1] = 10\ (10^{n-5}w[-1] + W[-1]) = \Gamma/100$, we may assume that $10w[-1]$ is the binary equivalent of decimal value $000\gamma_1\gamma_2$ (disregarding the fractional point). Also, recalling the 17-bit representation of $10w[i]$ (Section 3.1.1.2), we only need to convert decimal number $\gamma_1\gamma_2$ to its 2CLCS equivalent. This is achieved via carry-save addition of $8\gamma_1 + 2\gamma_1 + \gamma_2$.

*3.2.2 Precomputation of the divisor multiples:* Recall the practice of decomposing divisor $\Delta$ to two binary and decimal numbers $d$ and $D$ (see Section 3.1), such that $\Delta = 10^{n-5}d + D$. Let the four most significant fractional decimal digits of $\Delta$ be denoted as $\delta_1\delta_2\delta_3\delta_4 \equiv d$. Therefore disregarding the decimal point, $d = 1000\delta_1 + 100\delta_2 + 10\delta_3 + \delta_4$, which can be computed as in (19). Given that each $\delta$-digit is represented by a four-bit BCD number, (19) can be expressed in dot notation as in Fig. 9. Therefore a

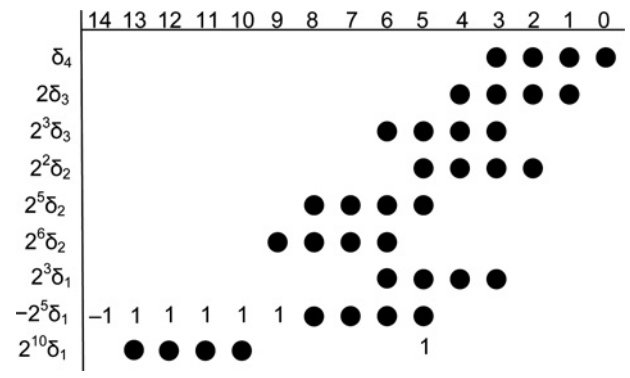


**Fig. 9** *Dot notation representation of d (unevaluated (19))*

three-level sparse CSA tree followed by an 11-bit adder is needed to compute $d$.

$$d = 2^{10}\delta_1 - 2^5\delta_1 + 2^3\delta_1 + 2^6\delta_2 + 2^5\delta_2 + 2^2\delta_2 + 2^3\delta_3 + 2\delta_3 + \delta_4 \quad (19)$$

The other $(n-5)$ BCD digits constitute $D$, the decimal component of $\Delta$. To compute the decimal components of positive divisor multiples (i.e. 2D, 3D, 4D, 5D), recall the well-known technique (e.g. [19, 22, 23]) where the 2 and 5 multiples of a decimal number can be generated via a simple logic in a carry-free manner, such that the latency is not dependent on the number of decimal digits. The rest of divisor multiples can be generated via implementation of $3D = 2D + D$ and $4D = 2 \times 2D$. The $D$-multiples are composed of $n-5$ BCD digits and carry-out digits described by Table 2. These carry bits (at most three) shall be fed into the corresponding CSA networks that compute the binary components of divisor multiples (i.e. $d$-multiples). The multiples $2d$ and $4d$ are simply achieved via one and two bit left shifts, thus making room for carries coming from the decimal parts. The 3- and 5-multiples can be computed as $3d = 2d + d$ and $5d = 4d + d$ via simple 17-bit adders.

To evaluate the negative multiples, we first compute the nine's (two's) complement of the corresponding decimal (binary) components of positive multiples. This will send a constant negabit $-1$ to position 0 of the binary part that cancels the post two's complement increment. However, the BCD CSA of PRC requires the ten's complement of the negative $D$-multiples, which will be accommodated by feeding the nine's complement $D$-multiples and an enforced carry-in to the rightmost full adder (the $s_q$ signal in Fig. 8). Recalling that $q_0 = 0$ and $q_1 \in \{0, 1\}$, the earliest that we

**Table 2** Decimal carries out of *D*-multiples

| Multiple | Range of carries | Representation |
|---|---|---|
| **2** | [0, 1] | ● |
| **3** | [0, 2] | ● ● |
| **4** | [0, 3] | ● ● ● |
| **5** | [0, 4] | ● ● ● |

need the divisor multiples is within the second recurrence cycle. Therefore the above computations will be off the critical path.

### 3.2.3 Generating the reduced comparison multiples

The comparison multiples $M_k$ were defined in Section 2.1 by (7), which is customised, for $\rho = 5/9$ as in (20), below.

$$(k - 5/9)d \le M_k \le (k - 1 + 5/9)d \tag{20}$$

To ease the computation of $M_k$, we define these multiples as in (21), but only compute them for $k > 0$, given that it is easy to see that $M_{-k} = -M_{k+1}$.

$$M_k = (k - 0.5)d; \quad k \in [-4, 5] \tag{21}$$

In Section 3.1, we explained the preference of using the reduced comparison multiples $m_k = M_k/10$. Therefore we first compute $d/10$ from $\delta_1\delta_2\delta_3$ (see Section 3.2.2) via a CSA network exactly as the top six rows of Fig. 9. Then, given that $d$, $2d$, …, $5d$ are already computed as binary parts of divisor multiples (Section 3.2.2), we compute $m_k$ as follows, where the 0.5, 2 and 4 multiples of $d/10$ are found using simple binary shift operations.

Recalling the discussion on the value of $q_1 \in \{0, 1\}$, in Section 3.2.1, only $m_1$ is needed in the first cycle. Therefore the operations in (22) do not fall within the critical path.

$$\begin{aligned} &m_1 = 0.5\frac{d}{10}, \quad m_2 = m_1 + \frac{d}{10}, \quad m_3 = m_1 + 2\frac{d}{10}, \\ &m_4 = 4\frac{d}{10} - m_1, \quad m_5 = 4\frac{d}{10} + m_1 \end{aligned} \tag{22}$$

Fig. 10 depicts the block diagram of the initialisation step.

### 3.3 Clean up

Four clean-up operations are needed to convert the minimally redundant decimal quotient (gained at the end of the $(n + 1)$th recurrence step) to the final BCD result, namely conversion to BCD, normalisation, correction and rounding.

The conversion and normalisation can be performed on-the-fly, and rounding increment does not affect any digit beyond $q_{n+1}$ [7]. The particular digit set that we use (i.e. $[-5, 5]$) does not introduce extra complexity to the techniques used in [7]. Rather, it leads to simpler conversion logic. When the last remainder is negative, the quotient should be corrected via a decrement. This does not need to actually take place, for the on-the-fly conversion always keeps the decremented quotient as well as the quotient. The IEEE 754-2008 standard for floating point arithmetic recommends the default RoundTiesToEven mode for decimal arithmetic. To implement such standard, one needs to perform an extra recurrence step to compute $q_{n+2}$, to be used for the rounding decision. (Recall that $q_{n+1}$ was shown to be required, in Section 3.2.1, owing to initialisation of $\Gamma/100$.) During this step, the previous quotient digit (i.e. $q_{n+1}$) is converted to BCD via on-the-fly conversion. One advantage of the digit set $[-5, 5]$ is that no rounding operation is needed except for the tie cases of $q_{n+2} \in \{-5, 5\}$. The reason is that for $q_{n+2} \in [0, 4]$, one just ignores $q_{n+2}$ and for $q_{n+2} \in [-4, -1]$, the conversion to BCD requires decrementing $Q$ and adding 10 to $q_{n+2}$. The latter results in $q_{n+2}$ to belong to [6, 9], which in turn requires incrementing $Q$. The net effect is that $q_{n+2}$ can be ignored and there needs to be no rounding increment. If we do the same in the tie cases of $q_{n+2} \in \{-5, 5\}$, the rounding error will be balanced if equal probability is assumed for the occurrence of $\pm 5$ digits. Therefore this so to say, auto-rounding complies with the goal of the RoundTiesToEven standard as to balancing the rounding errors. However, the actual BCD quotient may be 1 ulp off the standard result when the original $q_{n+1}$ is odd and $q_{n+2}$ is $\pm 5$.

### 3.4 Overview of the cycles

The cycling diagram of Fig. 11 describes how different operations within the proposed division scheme can be distributed in cycles. There are $n + 3$ cycles for producing an $n$-digit auto-rounded balanced-error quotient. However, an extra rounding cycle is required for 100% compliance with IEEE 754-2008 standard for RoundTiesToEven mode.

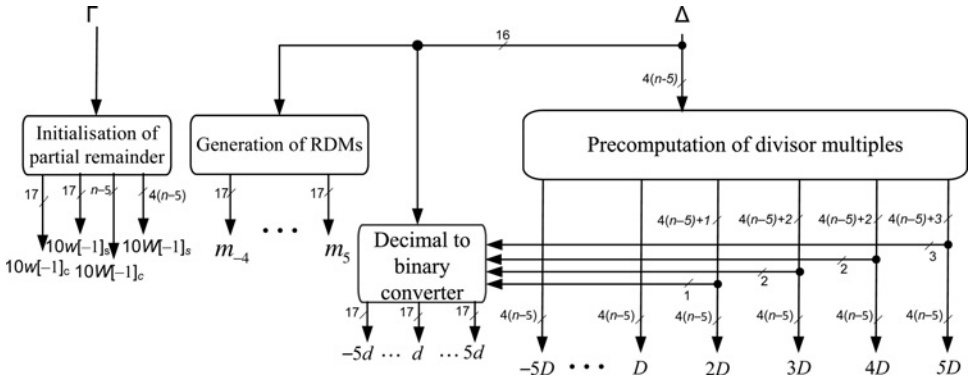


**Fig. 10** *Initialisation*




doi: 10.1049/iet-cdt.2010.0026

**Cycle 1 (Initialisation):**
1.1 Precompute the divisor multiples;
1.2 Generate the reduced comparison multiples;
$q_0 = 0$; $-q_0 \times d = 0$; $s_q = 0$;

**Cycle 2:**
Continue 1.1 and 1.2;
Select $q_1$ (Eqn. 17);
$w[0] = 10w[-1] - q_0 \times d$;
$W[0] = 10W[-1] - q_0 \times D$;

**Cycle 3:**
Continue 1.1;
Select $q_2$ (Eqn. 17);
$w[1] = 10w[0] - q_1 \times d$;
$W[1] = 10W[0] - q_1 \times D$;
On the fly conversion & normalisation of $q_1$;

**Cycle $i$ ($4 \leq i \leq n+2$) (Recurrence):**
Select $q_{i-1}$ (Eqn. 17);
$w[i-2] = 10w[i-3] - q_{i-2} \times d$;
$W[i-2] = 10W[i-3] - q_{i-2} \times D$;
On the fly conversion & normalisation of $q_{i-2}$;

| Auto-rounding | RoundTiesToEven |
|---|---|
| **Cycle $n$+3 (Clean Up):**<br>On the fly conversion & normalisation of $q_{n+1}$;<br>Correct the quotient if remainder is negative; | **Cycle $n$+3:**<br>Select $q_{n+2}$ (Eqn. 17);<br>On the fly conversion & normalisation of $q_{n+1}$;<br>**Cycle $n$+4 (Clean Up):**<br>RoundTiesToEven;<br>Correct quotient if remainder is negative; |

**Fig. 11** *Task distribution of the proposed divider in cycles*

## 4 Comparison with previous relevant works

In this section, we briefly describe the previous relevant works [6–8, 10] and compare them with the proposed divider. The latter two contributions, which are the latest reports on hardware decimal dividers that we have encountered, use LE analysis for performance evaluation and Vazquez *et al.* [8] have estimated the LE measures for [6, 7]. Therefore we also perform LE analysis on the proposed divider to estimate its latency and area.

In the LE analysis [24], latency is measured in FO4 units defined as the delay of an inverter with a fan-out of four inverters, and area consumption is evaluated in terms of minimum-sized NAND2 gate units. LE is ideal for evaluating alternatives in the early design stages and provides a good starting point for more intricate optimisations [24]. Therefore following [8], neither we undertake optimising techniques such as gate sizing, nor consider the effect of interconnections. Rather we allow gates with the drive strength of the minimum sized inverter and assume equal input and output loads. Furthermore, we overlook the latency of registers at the end of each cycle in all the studied dividers.

Given the wide spectrum of latency and area measures, as compiled in Table 4, the LE analyses is reliable for coarse evaluation. However, for more reliable and fine evaluation, and as a basis for more meaningful comparison of possible future works with the proposed one, we provide synthesis-based performance measures of the proposed divider.

### 4.1 Evaluation of the proposed decimal divider

The gate level analysis of the proposed divider, in Section 3.1.2, results in a reduced balanced-latency QDS (21 ΔG) and PRC (19 ΔG). This is confirmed by LE delay analysis, with 19.55 FO4 for QDS and 18.32 FO4 for PRC. Therefore we define the duration of each cycle as the equivalent of 19.55 FO4. The required cycles for the three steps of the division process is shown in Table 3, for $n = 16$, where auto-rounding is assumed. However, with the IEEE RoundTiesToEven, an extra cycle is needed for the recurrence step (i.e. 18 cycles).

The QDS in the proposed design is faster than [7, 8], for it is based on the best of the other two designs: namely, selecting the quotient digits from [−5, 5], as in the latter, and the binary manipulation of the few most significant digits of the partial remainders, as in the former. The former requires a more complex selector because of its wider digit set [−7, 7]. The latter, however, relies only on decimal manipulation of the partial remainders. Furthermore, the coder and selector functions, in [7, 8], are performed in two cycles (i.e. at the end of one cycle and the start of the next, respectively). However, we manage to do both in the same cycle and thus faster (Section 3.1.2).

The initialisation step, as was illustrated in Fig. 9, includes the computation of reduced comparison multiples $m_k$, where only $m_1$ is needed in the first cycle of the recurrence step. The rest of $m_k$s and $d$ are used from the second recurrence cycle on, whereas the other divisor multiples are not called for until the third recurrence cycle. Therefore the recurrence step can start as soon as $m_1$ is available. However, the computation of $m_1 = 0.5$ ($d$/10) includes conversion of the four most significant digits of divisor Δ to its binary equivalent, followed by a right-wired shift.

**Table 3** Number of required cycles for $n = 16$

| Division main step | Latency (cycles) |
|---|---|
| initialisation | 1 |
| recurrence | 17 |
| clean-up | 1 |
| total | 19 |

The process is similar to that explained in Section 3.2.2 and is composed of the function of two CSAs, followed by a seven-bit ripple-carry addition. The actual latency of such operations, using carry look-ahead logic, is equivalent to 14.98 FO4. Furthermore, $d$ can be made available at the start of the third cycle, since its computation (Section 3.2.2) requires 12.38 FO4 for three CSAs (Cycle 1) and 8.51 FO4 for 11-bit adder (Cycle 2). The other reduced comparison multiples (22), computed via 11-bit adders (8.51 FO4), will also be available before the second cycle ends. Consequently, dedication of only one cycle for initialisation is enough. The timing diagram in Fig. 12, as an illustration of the above discussion, shows how $m$, $d$ and $D$ multiples are available in due time.

The clean up includes on-the-fly conversion of the last quotient digit, normalisation and correction of the quotient and rounding increment in case of IEEE rounding. These can be done in the time equivalent to 10 FO4; hence, sufficiency of one cycle.

### 4.2 Decimal divider of [6]

Nikmehr *et al.* [6] presented a decimal digit-recurrence division algorithm with redundant quotient digits in $[-9, 9]$, represented by two equally weighted positive and negative BCD digits collectively called DSD. The partial remainders are also DSD numbers, but the comparison multiples, retrieved from a look-up table, are BCD numbers. Therefore they use a special redundant adder with one BCD and two DSD operands. This divider takes 19 cycles, each of 35.8 FO4 latency, for 16 digit operands.

### 4.3 Decimal divider of [7]

The digit recurrence division scheme of Lang and Nannarelli [7] uses decimal digit set $[-7, 7]$. A quotient digit is represented as $q = 5 \times q_H + q_L$, where $q_H \in \{-1, 0, 1\}$ and $q_L \in \{-2, -1, 0, 1, 2\}$. The main advantage of this representation is that it obviates the need for difficult divisor multiples 3, 6 and 7. Partial remainders are kept in BCD carry-save format and thus efficiently manipulated by BCD-CSAs. However, to attain high speed QDS, the partial remainders are represented in part in binary. The look-up table retrieval of comparison multiples and the on-the-fly conversion and normalisation of the quotient are in order. This divider takes 20 cycles, each of 33.10 FO4, for 16 digit decimal operands [8].

### 4.4 Decimal divider of [8]

Vazquez *et al.* [8] have presented a new decimal divider, where quotient digits belong to $[-5, 5]$ and partial remainders are non-redundant ten's complement decimal numbers with a 5-2-1-1 encoding of decimal digits [8]. They use a carry-propagating decimal adder in the PRC part, which remains out of the critical delay path for their 16-digit decimal divider. However, for wider decimal dividers (e.g. 34-digit operands recommended by IEEE 754-2008 standard), the PRC falls in the critical path. Therefore the cycle time for this divider depends on the width of the operands. The comparison multiples are generated in the initialisation cycle by means of combinational circuits. This divider, which consumes the least area among all, takes 21 cycles, each of 22.50 FO4, for 16-digit decimal operands.

### 4.5 Decimal divider of [10]

The digit-recurrence divider used in the decimal floating point unit of IBM Power6 is based on the pre-scaling technique of [11]. The redundant partial remainder version of this divider takes 48 cycles, each of 13 FO4, for 16-digit decimal operands. Resource saving is the main goal of this design. Nevertheless, there is no direct report on area consumption.

The area and delay comparison for the five described dividers is found in Table 4, where AutoR and IEEER stand for auto-rounding (Section 3.3) and IEEE RoundTiesToEven mode [25], respectively. The speed of the proposed IEEER (AutoR) divider is 21% (27%) more than that of the best previous method [8], but at a cost of 6% more area.

### 4.6 Synthesis-based comparison

For more realistic results, we produced VHDL code for the proposed 16-digit decimal divider and ran simulations and synthesis, using the Synopsis Design Compiler. The target library is based on typical TSMC 0.13 μm standard CMOS technology. The result showed 2.12 ns per cycle, overall latency of 40.28 ns (42.4 ns) and 41 494 (41 686) $\mu m^2$ of area for the proposed divider with auto rounding (IEEE compatible rounding). For a fair comparison (i.e. using the same technology and parameters) we decided to run the

**Table 4** Area and latency comparison for 16-digit decimal dividers

| Ratio | Area (NAND2) | Ratio | Latency (FO4) | No. of Cycles | Cycle time (FO4) | Divider | |
|---|---|---|---|---|---|---|---|
| 0.99 | 11 100 | 0.95 | 371.45 | 19 | 19.55 | AutoR | new |
| 1 | 11 130 | 1 | 391.00 | 20 | 19.55 | IEEER | |
| 0.94 | 10 500 | 1.21 | 472.50 | 21 | 22.50 | [8] | |
| 1.21 | 13 500 | 1.69 | 662.00 | 20 | 33.10 | [7] | |
| – | – | 1.59 | 624.00 | 48 | 13.00 | [10] | |
| 2.03 | 22 600 | 1.74 | 680.20 | 19 | 35.80 | [6] | |

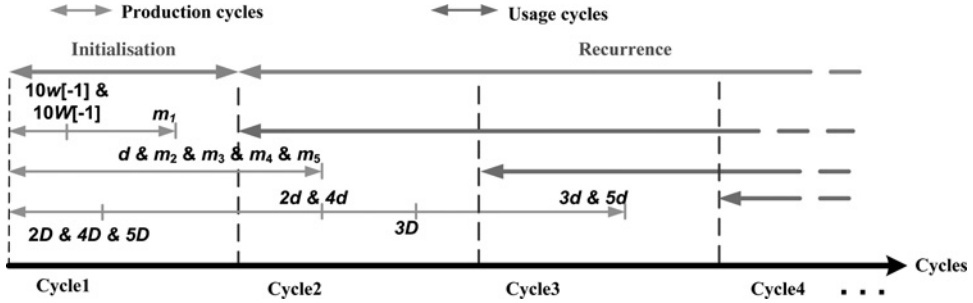


**Fig. 12** *Overlapping of the initialisation and recurrence cycles*




doi: 10.1049/iet-cdt.2010.0026

same procedure on the work of [7]. Furthermore, there are some black boxes in that design without adequate design information to enable us to synthesise what they had actually in mind (e.g. the coder and sign detection block). Therefore on the defined function of each black box, we tried to design its logic towards the best possible performance. The synthesis results turned out to be 3.91 ns per cycle, overall latency of 78.2 ns and 51 179 $\mu m^2$ of area. Consequently, the proposed decimal divider shows 46% less latency and 18.5% less area over that of [7]. The latter measures support the LE measures of 41 and 17.5% improvement for latency and area, respectively. One source of slower performance of [7] could be because of considerable more interconnections and wiring, caused by a larger digit set $[-7, 7]$.

## 5 Conclusions

We proposed a high-speed decimal divider with 21% more speed than the fastest previous relevant design [8] at the cost of 6% more area. Our design methodology is based on assembling redefined selected components of the dividers of two latest relevant works (i.e. [7, 8]). The selected components were modified (e.g. QDS selectors, replacing slow look-up tables with fast combinational logic) towards improved performance. Moreover, we used new components (e.g. QDS coder, binary multiplier by 10) in favour of enhanced overall performance. The particular design points of the proposed work can be highlighted as:

- Minimally redundant DSD (i.e. $[-5, 5]$) representation of the quotient;
- Reduced balanced-delay QDS and PRC;
- Effective one-cycle initialisation with precomputation of the divisor multiples and generation of comparison multiples without look-up tables;
- Binary/decimal partitioning of the carry-save partial remainders for faster PRC;
- Carry-free and fully-binary implementation of the high-speed QDS;
- Balanced error auto-rounding as a direct consequence of employing $[-5, 5]$ digit set leads to less number of cycles.

We briefly described other most recent relevant contributions (i.e. [6, 10]) and provided a comparison table (Table 4) with latency (FO4) and area (NAND2) measures. In addition, we synthesised the proposed work and that of [7] under the same technology. The latter reference work was considered for synthesis, since it was the only work with synthesis result. The speed of the auto-rounding version of the proposed design with one fewer cycle is 27% more than the fastest previous relevant design [8]. This design option, yet preserving the balanced error rounding, is appropriate for all special purpose applications that do not require 100% compliance with IEEE 754-2008 standard for floating point arithmetic [25].

Further research on this field can address similar designs for a decimal square rooter and combined divide/square root circuitry.

## 6 Acknowledgment

The authors wish to thank the anonymous reviewers for their valuable comments on the original manuscript. This research was supported in part by IPM under Grant CS1389-2-03, and in part by Shahid Beheshti Universtiy.

## 7 References

1 Parhami, B.: 'Computer arithmetic: algorithms and hardware designs' (Oxford University Press, 2010, 2nd edn.)
2 Robertson, J.E.: 'A new class of digital division methods', *IRE Trans. Electr. Comput.*, 1958, **EC-7**, (3), pp. 88–92
3 Robertson, J.E.: 'The correspondence between methods of digital division and multiplier recoding procedures', *IEEE Trans. Comput.*, 1970, **C-19**, (8), pp. 692–701
4 Montuschi, P., Ciminiera, L.: 'Over-redundant digit sets and the design of digit-by-digit division units', *IEEE Trans. Comput.*, 1994, **43**, (3), pp. 269–277
5 Wang, L.K., Schulte, M.J.: 'Decimal floating-point division using Newton–Raphson iteration'. Proc. IEEE Int. Conf. Application–Specific Systems, Architectures and Processors (ASAP), 2004, pp. 84–95
6 Nikmehr, H., Philips, B.: 'Fast decimal floating-point division', *IEEE Trans. Comput.*, 2006, **14**, (9), pp. 951–961
7 Lang, T., Nannarelli, A.: 'A Radix-10 Digit-recurrence division unit: algorithm and architecture', *IEEE Trans. Comput.*, 2007, **56**, (6), pp. 727–739
8 Vazquez, A., Antelo, E., Montuschi, P.: 'A radix-10 SRT divider based on alternative BCD codings'. XXV IEEE Int. Conf. on Computer Design (ICCD 2007), Lake Tahoe, CA, USA, 2007, pp. 280–287
9 Cowlishaw, M.F.: 'Decimal floating-point: algorism for computers'. Proc. 16th IEEE Symp. on Computer Arithmetic, June 2003, pp. 104–111
10 Schwarz, E.M., Carlough, S.R.: 'Power6 decimal divide'. Proc. IEEE Int. Conf. Application-Specific Systems, Architectures and Processors (ASAP), 2007, pp. 128–133
11 Ercegovac, M.D., Lang, T., Montuschi, P.: 'Very-high radix division with prescaling and selection by rounding', *IEEE Trans. Comput.*, 1994, **43**, (8), pp. 909–918
12 Kantabutra, V.: 'A recursive carry-lookahead/carry-select hybrid adder', *IEEE Trans. Comput.*, 1993, **42**, (12), pp. 1495–1499
13 Chaves, R., Sousa, L.: 'Improving residue number system multiplication with more balanced moduli sets and enhanced modular arithmetic structures', *IET Comput. Digit. Tech.*, 2007, **1**, (5), pp. 472–480
14 Han, D.G., Choi, D., Kim, H.: 'Improved computation of square roots in specific finite fields', *IEEE Trans. Comput.*, 2009, **58**, (2), pp. 188–196
15 Jaberipur, G., Kaivani, A.: 'Improving the speed of parallel decimal multiplication', *IEEE Trans. Comput.*, 2009, **58**, (1), pp. 1539–1552
16 Ercegovac, M.D., Lang, T.: 'Digital Arithmetic' (Morgan Kaufmann Publishers, 2004)
17 Nikmehr, H.: 'Architectures for floating-point division'. PhD dissertation, School of Electric and Electronic Engineering, University Adelaide, Adelaide, Australia, 2005
18 Ercegovac, M.D., Lang, T.: 'On-the-fly rounding', *IEEE Trans. Comput.*, 1992, **41**, (12), pp. 1497–1503
19 Richards, R.K.: 'Arithmetic operations in digital computers' (Van Nostrand, New York, 1955)
20 Schmookler, M., Weinberger, A.: 'High speed decimal addition', *IEEE Trans. Comput.*, 1971, **C-20**, (8), pp. 862–866
21 Jaberipur, G., Parhami, B.: 'Constant-time addition with hybrid-redundant numbers: theory and implementation', *Integr. VLSI J.*, 2008, **41**, (1), pp. 49–64
22 Erle, M.A., Schulte, M.J.: 'Decimal multiplication via carry-save addition'. Proc. IEEE Int. Conf. Applications–Specific Systems, Architectures and Processors (ASAP), 2003, pp. 348–359
23 Jaberipur, G., Kaivani, A.: 'Binary-coded decimal digit multipliers', *IET Comput. Digit. Tech.*, 2007, **1**, (4), pp. 377–381
24 Sutherland, I., Sproull, R., Harris, D.: 'Logical effort: designing fast CMOS circuits' (Morgan Kaufmann, 1999)
25 IEEE Standards Committee: '754-2008 IEEE standard for floating-point arithmetic', August 2008, pp. 1–58, doi: 10.1109/IEEESTD.2008.4610935

## 8 Appendix 1: determining the number of fractional digits

$$q_{i+1} = k, \quad \text{if } M_k \le 10\Omega[i] \le M_{k+1} \tag{23}$$

We call $t$ the number of fractional digits that is enough for QDS function. By applying the truncated numbers to inequality (23), we have

$$\{M_k\}_t \le \{10\Omega[i]\}_t < \{M_{k+1}\}_t \tag{24}$$

This interval is divided into

$$\{M_k\}_t \le \{10\Omega[i]\}_t \tag{25a}$$

$$\{10\Omega[i]\}_t < \{M_{k+1}\}_t \tag{25b}$$

$$M_k - 10^{-t} < \{M_k\}_t \le \{10\Omega[i]\}_t < 10\Omega[i] + 10^{-t} \tag{26}$$

or simply

$$M_k - 2 * 10^{-t} < 10\Omega[i] \tag{27}$$

Since $q_{i+1} = k$, adding $|k\Delta$ to inequality (27) and using (2) results in

$$M_k - 2 * 10^{-t} - k\Delta < \Omega[i+1] \tag{28}$$

In order to maintain the convergence condition, we obtain

$$-\rho\Delta \le M_k - 2 * 10^{-t} - k\Delta \Rightarrow (k-\rho)\Delta + 2 * 10^{-t} \le M_k \tag{29}$$

For interval (25b), the same derivation can be used to obtain

$$M_{k+1} \le (k+\rho)\Delta - 2 \times 10^{-t} \tag{30}$$

Replacing $k+1$ with $k$ in inequality (30) and combine it with inequality (29) leads to

$$(k-\rho)\Delta + 2 * 10^{-t} \le M_k \le (k-1+\rho)\Delta - 2 \times 10^{-t} \tag{31}$$

In order to make sure that finding a value for $M_k$ using (31) is always possible, inequality (32) should always be maintained.

$$(k-\rho)\Delta + 2 * 10^{-t} < (k-1+\rho)\Delta - 2 \times 10^{-t} \tag{32}$$

This leads to

$$10^t > 4/(2\rho - 1)\Delta \tag{33}$$

Inequality (33) can be used for determining the minimum number of truncated fractional digits needed for correct QDS thus

$$t = 1 + \left\lceil \log_{10}\left(\frac{36}{2\alpha - 9}\right)\right\rceil \tag{34}$$

## 9 Appendix 2: details of the width compressor in Part V of Fig. 6

Fig. 13 depicts a symbolic view of the value-preserving transformation of the leftmost 11-bit collection, of the dashed box in Part V, to the two equally weighted negabits of Part VI.

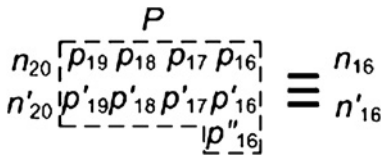


**Fig. 13** *Symbolic view of the width compression*

The reduction process, which can actually start at the beginning of Part IV, is based on checking the conditions under which the collective value of the left bit-collection falls within the range $[-2, 0]$, as is required by (16), where $e = P - 16(n_{20} + n'_{20})$. Table 5 is a compact truth table that helps in deriving the logical equation for $n_{16}$ and $n'_{16}$.

The derivation can be simplified by checking the carry out of addition of the three components of posibit-collection $P$ (i.e. the dashed box), represented as $(P \ge 16)$, and that of $P+1$, represented as $(P \ge 15)$, against the collective value of the two negabits $n_{20}$ and $n'_{20}$. The following equations represent a simple solution to the latter, where $G_{i:j}$ is the group-generate signal, $P_{i:j}$ is group-propagate signal and $g_i = p_i \wedge p'_i$ is the $i$th generate signal for $(16 \le i, j \le 19)$.

$$n_{16} = (n_{20} \wedge n'_{20}) \vee [(n_{20} \vee n'_{20}) \wedge \overline{(P \ge 16)}]$$

$$n'_{16} = (n_{20} \wedge n'_{20} \wedge \overline{(P = 31)}) \vee [(n_{20} \vee n'_{20}) \wedge \overline{(P \ge 15)}]$$

$$(P = 31) = g_{19} \wedge g_{18} \wedge g_{17} \wedge g_{16} \wedge p''_{16}$$

$$(P \ge 16) = G_{19:17} \vee P_{19:17} \wedge [g_{16} \vee (p_{16} \wedge p''_{16}) \vee (p'_{16} \wedge p''_{16})]$$

$$(P \ge 15) = (P+1) \ge 16 = G_{19:17} \vee P_{19:17} \wedge [p_{16} \vee p''_{16} \vee (P_{19:18} \wedge g_{16} \wedge p''_{16})]$$

**Table 5** Compact truth table for derivation of $n_{16}$, $n'_{16}$

| $n_{20}$ | $n'_{20}$ | $16(n_{20} + n'_{20})$ | Valid values for $P$ | $e = n_{16} + n'_{16}$ | $(n_{16}, n'_{16})$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | (0, 0) |
| 0 | 1 | −16 | 14, 15, 16 | −2, −1, 0 | (1, 1), (1, 0), (0, 0) |
| 1 | 0 | −16 | 14, 15, 16 | −2, −1, 0 | (1, 1), (1, 0), (0, 0) |
| 1 | 1 | −32 | 30, 31 | −2, −1 | (1, 1), (1, 0) |




doi: 10.1049/iet-cdt.2010.0026