Design of a fast radix-4 SRT divider and its VLSI implementation

C.-L.Wey and C.-P.Wang

Abstract: The design of a fast divider is an important issue in high-speed computing. The paper presents a fast radix-4 SRT division architecture. Instead of finding the correct quotient digit, an estimated quotient digit is first speculated. The speculated quotient digit is used to simultaneously compute the two possible partial remainders for the next step while the quotient digit is being corrected. Thus, this two-step process does not influence the overall speed. Since the decision-making circuits can be implemented with simple gate structures, the proposed divider offers fast speed operation. Based on the physical layout, the circuit takes 247ns for a double precision division (56 bits for fraction part), where the 2 μm CMOS technology in MAGIC is employed and simulated.

1 Introduction

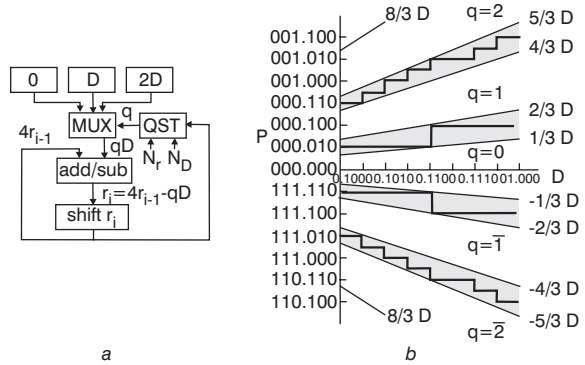
The design of fast dividers is an important issue in high-speed computing because division accounts for a significant fraction of the total arithmetic operation [1]. Most implementations for the division are based on the SRT algorithm that uses a recurrence producing one quotient digit for each step [2–9]. The speed of such SRT-based dividers is mainly determined by the complexity of the quotient-digit selection. Fig. 1 illustrates an architecture of a radix-4 SRT divider which employs a quotient-digit selection table (QST). The use of QST significantly reduces the complexity of quotient-digit selection. However, the table size increases drastically with high radices [2, 9]. The table size can be reduced significantly by estimating the quotient digit instead of finding the exact one [9]. The estimated quotient digit is calibrated in parallel with updating the new partial remainder. Since the two-step process does not affect the division speed, the approach has fast speed performance due to the significant reduction in table size. This paper presents the detailed design of a fast radix-4 SRT division and its VLSI implementation. Results show that, based on the physical layout, the circuit takes 291ns to compute a double precision division (56 bits in the fraction part), where the CMOS technology in MAGIC is employed for simulation.

2 Radix-4 division

Consider the following recursive equation for the partial remainder in a high-radix SRT division [2], $r_i = \beta_{i-1} - q_i D$, where $\beta = 2^m$ is the radix, D is the divisor, r_{i-1} is the partial remainder at the $(i-1)$th step, and q_i is the ith quotient digit. The high-radix SRT division can be implemented in such a way that its quotient digit q_i is selected from a digit set $\{-\alpha, \ldots, -1, 0, 1, \ldots, \alpha\}$, where $\lceil(\beta-1)/2\rceil \le \alpha \le (\beta-1)$. The ratio $\kappa = \alpha/(\beta-1)$ is a measure of the redundancy in the representation of the quotient digits. The smaller the value of κ, the smaller is the redundancy in the number system for the quotient.

Fig. 1*b* shows the $P-D$ plot of a radix-4 division [2], where $(\beta,\alpha)=(4,2)$, $\kappa = 2/3$ and $D = [D_{\min}, D_{\max}) = [0.5, 1.0)$. The quotient digit $q \in \{-2,-1,0,1,2\}$. Let $P = 4r_{i-1}$ denote the previous partial remainder, where r_{i-1} is the partial remainder at the $(i-1)$th step and $|r_{i-1}| \le (2/3)D$. Thus, the upper limits and lower limits for P are

$$(-2/3+q)D \le P \le (2/3+q)D \tag{1}$$



q	Partial remainder P
2	(4/3) $D \le P \le$ (8/3) D
1	(1/3) $D \le P \le$ (5/3) D
0	(−2/3) $D \le P \le$ (2/3) D

q	Partial remainder P
−2	(−8/3) $D \le P \le$ (−4/3) D
−1	(−5/3) $D \le P \le$ (−1/3) D

c

	Partial remainder P
$P_{1,2}$	(4/3) $D \le P \le$ (5/3) D
$P_{0,1}$	(1/3) $D \le P \le$ (2/3) D

	Partial remainder P
$P_{-2,-1}$	(−5/3) $D \le P \le$ (−4/3) D
$P_{-1,0}$	(−2/3) $D \le P \le$ (−1/3) D

d

Fig. 1 *Radix-4 SRT division*

a Architecture; *b* *P-D* plot and stair function for $(\beta,\alpha)=(4,2)$, where shaded areas are overlapping regions; *c* Regions for various partial remainders P; *d* Overlapping regions

IEE Proceedings online no. 19990524

DOI: 10.1049/ip-cdt:19990524

Paper first received 13th November 1997, and in revised form 21st April 1999

The authors are with the Department of Electrical Engineering, Michigan State University, East Lancing, MI 48824-1226, USA

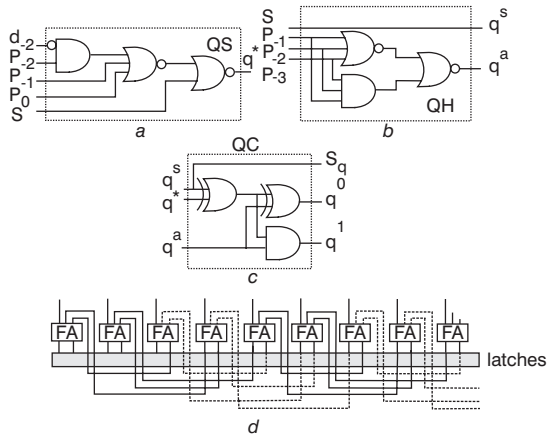


Fig. 5 *Radix-4 SRT divider*
a Block QS; *b* Block QH; *c* Block QC; *d* CSA with shifter

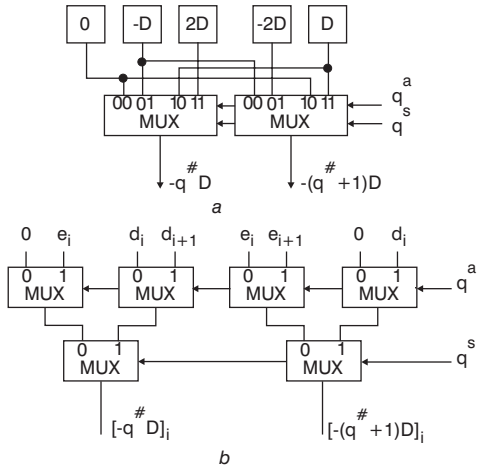


Fig. 6 *Radix-4 SRT divider*
a, b MUXC implementation

Table 2: Comparison Constants for $q^{\#}$

$SP_{-1}P_{-2}.P_{-3}$	$q^{\#}$	q^s	q^a
011.1	x	x	x
01x.x	1	0	1
001.x	1	0	1
000.1	1	0	1
000.0	0	0	0
111.1	−1	1	0
111.0	−2	1	1
110.x	−2	1	1
10x.x	−2	1	1
100.0	x	x	x

bits, we obtain (SP_{-1} P_{-2} P_{-3}) which is used to compare the constants given in Table 2. According to the encoding scheme in Table 1, eqn. 5 implies that $q^s = S$ and $q^a = 0$ if $(P_{-1}\ P_{-2}\ P_{-3}) = (00.0)$ or (11.1), where q^a is expressed by the following logic function:

$$q^a = [P_{-1}P_{-2}P_{-3} + P'_{-1}P'_{-2}P'_{-3}]' \quad (9)$$

Fig. 5*b* shows the logic implementation of eqn. 9.

3.2.3 Block QC: Block QC sums up the estimated quotient digit $q^{\#}$ and the correction value q^* to produce the actual ith quotient digit q_i, i.e. $q_i = q^{\#} + q^*$, where $q_i \in \{-2, -1, 0, 1, 2\}$. Let $q = (S_q q^1 q^0)$ be in sign-magnitude form, where S_q is the sign bit and $(q^1 q^0)$ represents the binary representation of the absolute value of q. Table 3 lists the truth table for $q = (S_q q^1 q^0)$ in terms of q^s, q^a and q^*. The following logic functions realise the truth table

$$S_q = q^s,\ q^1 = q^a(q^s \oplus q^*) \text{ and } q^0 = q^a \oplus (q^s \oplus q^*) \quad (10)$$

Fig. 5*c* shows the logic implementation of eqn. 10.

Table 3: Truth table for block OC

			$q^* = 0$				$q^* = 1$			
$q^{\#}$	q^s	q^a	q	S^q	q^1	q^0	q	S^q	q^1	q^0
1	0	1	1	0	0	1	2	0	1	0
0	0	0	0	0	0	0	1	0	0	1
−1	1	0	−1	1	0	1	−0	1	0	0
−2	1	1	−2	1	1	0	−1	1	0	1

3.2.4 Adders–CSA and CLA: Two CSAs (carry-slave adders) are used to compute the remainders P^0 and P^1. The CSA with shifting is realised by FAs and latches, as shown in Fig. 5*d*. The sum and carry outputs of the first seven FAs are fed to a 7-bit CLA.

3.2.5 Multiplexer circuitry: The multiplexer circuitry (MUXC), as shown in Fig. 6 generates $-q^{\#}D$ and $-(q^{\#}+1)D$, which are chosen from $\{-2D, -D, 0, D, 2D\}$. Since the divisor D is an n-bit positive normalised binary number, the numbers $-D$ and $-2D$ are represented in two's complement form. More specifically, let D be denoted as $(S^d d_0.d_{-1}d_{-2}\ldots d_{-n})$, where $S^d = 0$ is the sign bit, $d_0 = 0$ and $d_{-1} = 1$. Then, $-D = ((S\text{-}d)'e_0.e_{-1}e_{-2}\ldots e_{-n})$ and $-2D = ((S^d)'e_{-1}e_{-2}\ \ldots\ e_{-n}0)$, where $e_i = d_i'$ and $e_{-n} = d_{-n}' + 1$. Note that the $(-n)$th bit of $(-2D)$, or, namely, $e_{-(n+1)}$, is a 0. The two's complementation is realised by using a simple one's complementation and assigning a 1 as the initial carry of the CSA.

The estimated quotient digit, $q^{\#} \in \{1, 0, -1, -2\}$, is represented by two bits, i.e. $q^{\#} = (q^s, q^a)$, and its values, 1, 0, −1 and −2, are encoded as (0, 1), (0, 0), (1, 0) and (1, 1), respectively, as shown in Table 4. Fig. 6*a* illustrates the block diagram that implements the MUXC. Each bit slice of the MUXC is realised by six 2-to-1 multiplexers (MUXs), as shown in Fig. 6*b*, where the first level of MUXs is selected by the signal q^a, while the second level is determined by q^s. The output of the ith bit slice in MUXC is fed to the CSA, as shown in Fig. 7*a*. To generate the initial carry of CSA, the inputs to the $-n$th bit of

Table 4: Function MUXC and bit assignment for (-n) bit of CSA

$q^{\#}$	q^s	q^a	$-q^{\#}D$	y_{-n}	$-(q^{\#}+1)D$	y_{-n}
1	0	1	$-D$	1	$-2D$	1
0	0	0	0	0	$-D$	1
−1	1	0	D	0	0	0
−2	1	1	$2D$	0	D	0

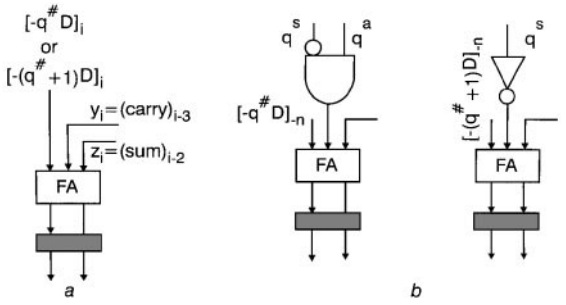


Fig. 7 *Radix-4 SRT divider*
a Inputs to *i*th bit of CSA
b Inputs of -*n*th bit of CSA

MUXC are modified by assigning a 0 to $d_{i+1}=d_{-(n+1)}$ and a 1 to $e_{i+1}=e_{-(n+1)}$. The inputs to the last bit of CSA are modified as follows: the initial carry of CSA is fed to y_{-n} of the $-n$th FA, where $y_{-n}=0$ (1) if the CSA is used as an adder (subtracter). Therefore, for $-q^{\#}D$, $y_{-n}=0$ for 0, D and $2D$, and $y_{-n}=1$ for $-D$, as shown in Table 4. On the other hand, for $-(q^{\#}+1)D$, $y_{-n}=0$ for 0 and D, and $y_{-n}=1$ for $-D$. However, for $-2D$, it is necessary to add the initial carry '1' to the carry-in bit of the $[-(n-1)]$th bit slice. Without modifying the $[-(n-1)]$th bit slice, this can be achieved by setting $y_{-n}=1$ and $[-(q^{\#}+1)D]_{-n}=1$ to the inputs of the CSA in Fig. 7*b*. Since $e_{-(n+1)}=0$ for $-2D$, thus assigning $[-(q^{\#}+1)D]_{-n}=1$ is equivalent to setting the input $e_{-(n+1)}$ to a 1, as shown inFig. 6*a*. Therefore, the logic function for y_{-n} is $(q^s)'\ q^a$ and $(q^s)'$ for $-q^{\#}D$ and $-(q^{\#}+1)D$, respectively.

3.2.6 VLSI implementation–speed and area: Fig. 8 shows the physical layout of an appropriate 56-bit floating-point divider (fraction part). The layout is generated by the MAGIC layout editor, where 2 μm SCMOS technology is employed. The layout includes a 57-bit CSA, two 7-bit CLAs with 1-level, a 57-bit MUXC with 4-to-1 MUXs, and BLOCK QS, QC and QH. There, different types of MUX circuits are used. Type- 1, MUX^1, includes the one receiving its inputs from CLAs and the one sending its outputs to QH; type-2, MUX^2, includes the one receiving its inputs from CSAs and the one sending the outputs to CSAs; and type-3, MUX^3, is the MUXC. The propagation delay time of each unit has been simulated by *Pspice,* where the circuit parameters are extracted from the layout. Table 5 lists the size and propagation delay of each unit. According to the layout in Fig. 8 with the routing areas, for

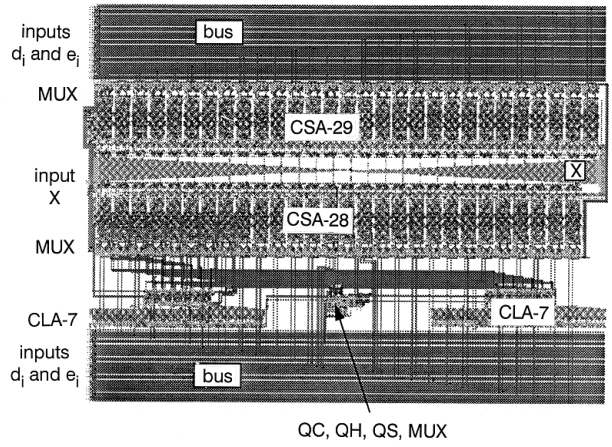


Fig. 8 *Physical layout of radix-4 SRT divider*

Table 5: Simulation results for radix-4 SRT divider

Unit	Size	Delay, ns
CSA	0.02485 mm^2/bit	3.8
CLA-7	0.446 mm^2	3.8
QC	0.0148 mm^2	3.8
QH	0.008 mm^2	2.0
QS	0.005 mm^2	3.5
MUX^1	0.008 mm^2	0.25
MUX^2	0.007493 mm^2/bit	0.25
MUX^3	0.0138 mm^2/bit	0.43

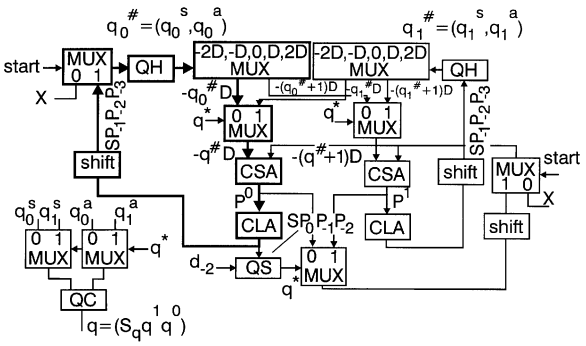


Fig. 9 *Schematic diagram of improved version with extra MUXC*

56-bit radix-4 SRT division, the total area is approximately 3.7×5.3 mm^2, and its delay time is 13.9 ns per cycle.

The division time can be improved by the alternative architecture shown in Fig. 9 where an extra MUXC is used. As indicated by the bold lines, Block QS is no longer in the critical path. Thus, the critical path includes the MUX^3, MUX^2, CSA, CLA, MUX^1 and QH. Simulation results show that the circuit has a delay of 10.4 ns per cycle, or 291.2 ns for the 56-bit division. However, the speed improvement is achieved at the increased cost of area which is nearly 4×5.3 mm^2, as shown in Fig. 10.

Since block QH is on the critical path, improving its speed performance will also make the division process faster. Eqn. 9 shows that the output q^a is a function of P_{-1}, P_{-2} and P_{-3}. Therefore, the output q^a, as a function of three variables, can be realised by a 4-to-1 MUX which takes only 0.43 ns. As a result, the total delay in the critical path is 8.83 ns. Therefore, the proposed SRT algorithm can be achieved with a delay of 247.24 ns for 56-bit division, where 2 μm CMOS technology is employed.

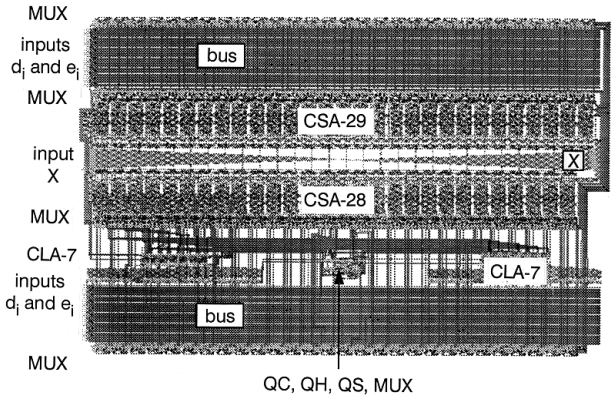


Fig. 10 *Physical layout of improved version*

Table 6: Double precision division performance comparison

Chip	Fraction only time, ns	Radix
W3364 (weitek)	675	4
R3010B (MIPS)	560	4
Method-A [10]	352	8
Method-B [10]	238	8
Self-timed [11]	160	2

Table 6 shows the performance comparison given in [10], where 1.2 μm CMOS technology is employed for all cases. Comparing the proposed radix-4 division, 247.24 ns with 2 μm CMOS technology, to the performance of various designs shown above, the proposed circuit performance is promising. The proposed circuit can be further improved by using 1.2 μm or better technology.

4 Conclusion

This paper presents a simple yet fast radix-4 divider design and its VLSI implementation. The proposed division method first speculates a quotient digit. The speculated digit is used to compute the two possible partial remainders, for the next step, in parallel with the quotient-digit correction process. The algorithm can be implemented with only a delay of 247.24 ns for double precision division. Although this paper is presented only for radix-4 division, the same design concept is readily extended for high radices to reduce the quotient-digit selection table size and make the use of high-radices to become possible and practical [9]. Also, similarly to the discussion in [10], the proposed architecture can also be implemented for square-root calculation. It should be mentioned that the primary focus of the developed algorithm and hardware implementation was placed on optimising execution. It is worthwhile to further investigate and develop an alternative design and implementation which is optimised in terms of execution time, chip area and/or power dissipation.

5 References

1 BOSE, B.K., PEI, L., TAYLOR, G.S., and PATTERSON, D.A.: 'Fast multiply and divide for VLSI floating-point unit', Proc. IEEE Symp. on *Computer Arithmetic*, Como, Italy, 1987, pp. 87–94

2 KOREN, I.: 'Computer arithmetic algorithms' (Prentice-Hall, Englewood Cliffs, NJ, 1993)

3 ERCEGOVAC, M.D., and LANG, T.: 'Simple radix-4 division with operands scaling', *IEEE Trans. Comput.*, 1990, **C-39**, pp. 1204–1207

4 ERCEGOVAC, M.D., and LANG, T.: 'Digital-recurrence algorithms and implementations for division and square root' (Kluwer Academic Publishing, 1993)

5 KRISHNAMURTHY, E.V.: 'On range-transformation techniques for division', *IEEE Trans. Comput.*, 1970, **C-19**, pp. 157–160

6 ERCEGOVAC, M.D., and LANG, T.: 'A division algorithm with prediction of quotient digits', Proc. 7th IEEE Symposium on *Computer Arithmetic*, Urbana, IL, 1985, pp. 51–56

7 FANDRIANTO, J.: 'Algorithm for high speed shared radix 4 division and radix 4 square root', Proc. 8th IEEE Symp. on *Computer Arithmetic*, Como, Italy, 1987, pp. 73–79

8 MONTUSCHI, P., and LUIGI CIMINIERA, 'Design of a radix 4 division unit with simple selection table', *IEEE Trans. Comput.*, 1992, **41**, (12), pp. 1606–1611

9 PAN, T.-H., KAY, H.-S., CHUN, Y., and WEY, C.L.: 'High-radix SRT division with speculation of quotient digits', Proc. International Conference of *Computer Design* (ICCD), Austin, Texas, 1995, pp. 479–482

10 HOBSON, R.F., and FRASER, M.W.: 'An efficient maximum-redundancy radix-8 SRT division and square-root method', *IEEE J. Solid-State Circuits*, 1995, **30**, (1), pp. 29–38

11 WILLIAMS, T.E., and HOROWITZ, M.A.: 'A zero-overhead self-timed 160ns 54-bit CMOS divider', *IEEE. Solid-State Circuits*, 1991, **26**, (11), pp. 1651–1661

6 Appendix: Error analysis

This Appendix is to justify that a 7-bit CLA is sufficient for this implementation. Let C_j and S_j, $j = s, 1, 0, -1, \ldots, -n$, denote the jth carry and sum bits of the CSA for the remainder P^0 at the ith cycle, respectively. Generating the complete $(n+3)$ bit binary value for $P^0 = (SP_1\ P_0.\ P_{-1}\ P_{-2} \ldots P_{-n})$ requires an $(n+3)$-bit CLA to sum the carry and sum bits as follows:

$$\begin{array}{ll} Ss\ S_1\ S_0\ S_{-1}\ S_{-2}\ S_{-3}\ S_{-4}\ S_{-5}\ \ldots\ S_{-(n-1)} & S_{-n} \\ +C_1\ C_0\ C_{-1}\ C_{-2}\ C_{-3}\ C_{-4}\ C_{-5}\ C_{-6}\ \ldots\ C_{-n} & 0 \\ \hline S\ P_1\ P_0\ P_{-1}\ P_{-2}\ P_{-3}\ P_{-4}\ P_{-5}\ \ldots\ P_{-(n-1)} & P_{-n} \end{array}$$

To avoid long carry propagation, this implementation rounds the value of P^0 to the bit P_{-t}, i.e

$$\begin{array}{l} Ss\ S_1\ S_0\ S_{-1}\ S_{-2}\ S_{-3}\ \ldots\ S_{-t} \\ +C_1\ C_0\ C_{-1}\ C_{-2}\ C_{-3}\ C_{-4}\ \ldots\ C_{-(t+1)^*} \\ \hline S\ P_1\ P_0\ P_{-1}\ P_{-2}\ P_{-3}\ \ldots\ P_{-t} \end{array}$$

where $C_{-(t+1)}{}^* = C_{-(t+1)} + C^{in}$. The rounding scheme defines the value of C^{in}, where $C^{in} = 0$ if $C_{-(t+2)} = S_{-(t+1)} = 0$, and $C^{in} = 1$, otherwise. The signal C_{in} is ORing both $S_{-(t+1)}$ and $C_{-(t+2)}$. Note that P_1 is not connected to block QH nor block QS.

Let P^0_{ro} and P^0_{tr} denote the rounded and truncated values of P^0. If $C^{in} = 0$, then $P^0_{ro} = P^0_{tr}$ and thus no error results. On the other hand, if $C^{in} = 1$, i.e. $P^0_{ro} \neq P^0_{tr}$, then two cases can be identified: both values result in either the same $q^{\#}_i$ and $q_i{}^*$, or different $q^{\#}_i$ and $q_i{}^*$, where $q^{\#}_i$ and $q_i{}^*$ are the ith estimated quotient digit and the corrected value, respectively. More specifically, in the former case, both values may be different, but they may result in the same $q^{\#}_i$ and $q_i{}^*$. Thus, both values generate the same quotient digit q and no error results. For the latter case, both values may be located at the different regions in Fig. 2, and P^0_{tr} and P^0_{ro} result in $q^{\#}_i = w$ and $q^{\#}_i = w + 1$, respectively. If the real remainder lies in the overlapping regions $P_{0,1}$, $P_{-1,0}$ or $P_{-2,-1}$, then we should be able to generate $q^* = 1$ and 0 for the truncated and rounded remainder, respectively. This results in both having the same quotient digit, i.e. $q = w + 1$. Let e_{max} be the maximum error which causes both truncated and rounded values to be located in the different $q^{\#}$-regions, but they are still in the same overlapping region. For the overlapping region $P_{0,1}$, we have $\varepsilon_{max} = P^0_{ro} - P^0_{tr}$. or $P^0_{tr} = P^0_{ro} - \varepsilon$, and $(1/3)D_{max} \leq P^0_{tr} = P^0_{ro} - \varepsilon_{max} = c_1 - \varepsilon_{max}$. Therefore, $\varepsilon_{max} = c_1 - (1/3) = 0.50_{10} - 0.33_{10} = 0.17_{10} > 0.125_{10} = 0.001_2$, i.e. $P^0_{tr} = c_1 - \mathrm{e}_{max} > 0.011_2$. This confirms that the first seven significant digits, $(SP_1\ P_0.\ P_{-1}\ P_{-2}\ P_{-3}\ P_{-4})$, are sufficient in this implementation.