A Combined Decimal and Binary Floating-point Divider

Sonia González-Navarro
University of Málaga
sonia@ac.uma.es

Alberto Nannarelli
Technical University of Denmark
an@imm.dtu.dk

Michael Schulte
University of Wisconsin-Madison
schulte@engr.wisc.edu

Charles Tsen
Nvidia Corporation
ctsen@nvidia.com

***Abstract*—In this paper, we present the hardware design of a combined decimal and binary floating-point divider, based on specifications in the IEEE 754-2008 Standard for Floating-point Arithmetic. In contrast to most recent decimal divider designs, which are based on the Binary Coded Decimal (BCD) encoding, our divider operates on either 64-bit binary encoded decimal floating-point (DFP) numbers or 64-bit binary floating-point (BFP) numbers. The division approach implemented in our design is based on a digit-recurrence algorithm. We describe the hardware resources shared between the two floating-point datatypes and demonstrate that hardware sharing is advantageous. Compared to a standalone DFP divider, the combined divider has the same worst case delay and 17% more area.**

I. Introduction

Decimal floating-point (DFP) research and hardware implementations of decimal arithmetic units have gained importance in recent years. DFP representations provide better accuracy in commercial and financial applications than binary floating-point (BFP) units, because BFP representations cannot accurately represent many fractional decimal numbers [1]. Furthermore, correct decimal rounding is required in several commercial and financial applications. In 2008, the IEEE 754 Standard for Binary Floating-point Arithmetic was revised and specifications for DFP formats and operations were added [2]. In the revised IEEE 754-2008 Standard, significands of DFP numbers can be represented with either the Densely-Packed Decimal (DPD) encoding or the Binary Integer Decimal (BID) encoding. DPD is a compressed form of the Binary Coded Decimal (BCD) encoding. With the BID encoding, also known as the binary encoding of DFP numbers, the significand of a DFP number is encoded as an unsigned binary integer. For example, the DFP number 0.105 is represented in BID by 105×10^{-3}, with significand $0 \ldots 001101001_2$ and exponent $-3 + bias$.

Although most recent decimal divider designs are based on the BCD encoding [3],[4],[5], using the DPD encoding for floating-point and the BCD encoding for fixed-point, this work proposes a combined DFP and BFP unit for division operating on BID-encoded DFP numbers. Since the BID encoding represents significands as unsigned binary integers, it seems appropriate to combine both BID and BFP in the same arithmetic unit. A combined BID and BFP unit has been proposed for multiplication [6]. Combined binary and BCD units have been proposed for addition [7], multiplication [8], [9] and division [10].

This work was supported in part by the Ministry of Education of Spain under contract TIN2006-01078. Charles Tsen contributed to this work while a student at the University of Wisconsin - Madison.

The combined unit we propose operates on either 64-bit BID-encoded DFP numbers or 64-bit BFP numbers and it is based on the radix-10 division unit presented in [11]. This unit is implemented using the digit-recurrence approach [12] and it has been modified to also support 64-bit BFP numbers. The BFP division is implemented with a retimed radix-16 (two overlapped radix-4 stages) digit-recurrence unit with selection by comparison, as in [10].

We show that adding BFP support for division in the BID divider of [11] does not affect the delay and has a small impact on the total area. Contributions of this paper include (1) providing the first algorithm and hardware design for a combined BFP and BID division unit, (2) providing area and delay estimates through synthesis of the proposed design, and (3) comparing the area and delay estimates of the proposed combined division unit to the standalone BID division unit and the combined DPD/BFP unit from [10].

The remainder of this paper is organized as follows: Section II describes the BFP and BID datatypes that our design supports. Section III introduces our combined BFP and BID digit-recurrence division algorithm. Section IV summarizes the division unit design and analyzes hardware sharing potential in the combined BFP and BID division unit. Section V presents and discusses our synthesis results. Section VI concludes the paper.

II. IEEE 754-2008 formats

The IEEE 754-2008 Standard, includes four formats to represent BFP numbers (*binary16, binary32, binary64* and *binary128*), and three formats (*decimal32, decimal64* and *decimal128*) to represent DFP numbers. The BFP and DFP number formats use three fields to define a number: a sign, an exponent, and a significand. The value of a finite BFP/DFP number is: $(-1)^s \times C \times b^{e-bias}$, where s is the sign, C is the significand, b is the base ($b = 2$ for BFP and $b = 10$ for DFP numbers), e is the biased exponent, and $bias$ is a positive constant which ensures that the stored exponent is non-negative. In the case of BFP numbers, *C* is normalized, and its most significant bit is implicitly 1. However, in the case of DFP the significand *C* is not normalized. Table I shows the parameter values of the 64-bit formats supported by our division unit.

TABLE I
64-BIT FLOATING-POINT FORMAT PARAMETERS

Format	Precision	Exponent length (bits)	Exponent bias
binary64	53 bits	11	1023
decimal64	16 digits	10	398

III. Combined Floating-point Division Algorithm

In this section, we introduce the combined BID/BFP divider and summarize the theory of the radix-r division digit-recurrence algorithm.

The BID/BFP division algorithm consists of the following steps:

Step 1: If the inputs operand are DFP numbers, normalize their significands.

Step 2: Divide significands, compute the sign, and subtract the exponents to obtain the intermediate exponent.

Step 3: If needed, normalize and round the quotient and update the exponent.

The division of the significands in Step 2 is implemented using a digit-recurrence algorithm [12]. Therefore, in order to apply this algorithm, the divisor has to be normalized. It is also convenient to normalize the dividend to help guarantee convergence and to reduce the number of leading zeros in the quotient. Therefore, both operands are normalized before dividing significands. Since BFP numbers are already normalized, Step 1 is applied only for DFP numbers. The normalization for BID-encoded DFP numbers is explained in detail in [11]. The algorithm is completed by converting the quotient from signed-digit to the required representation and rounding.

We now summarize the theory of the radix-r digit-recurrence algorithm assuming that the divisor and dividend are normalized. The division $q = x/d$ is implemented by the radix-r digit-recurrence iteration [12]

$$w[j+1] = rw[j] - q_{j+1}d \qquad j = 0, 1, 2, \ldots \tag{1}$$

where d is the divisor and $w[j]$ is the residual at iteration j.

The quotient-digit q_{j+1} is computed at each iteration by a selection function

$$q_{j+1} = SEL(\hat{d}, \widehat{rw[j]})$$

where $\hat{d}$ and $\widehat{rw[j]}$ are estimates of the divisor and the residual, respectively.

To obtain simpler selection functions we use a redundant digit set [12]. Moreover, the quotient-digit is split into two parts q_H and q_L such that

$$q_{j+1} = kq_{Hj+1} + q_{Lj+1}$$

where $k = 5$ for $r = 10$ and $k = 4$ for $r = 16$, and the digit sets for radix-16 are $q_H = \{-2, -1, 0, 1, 2\}$ and $q_L = \{-2, -1, 0, 1, 2\}$, and for radix-10 are $q_H = \{-1, 0, 1\}$ and $q_L = \{-2, -1, 0, 1, 2\}$.

By the quotient-digit decomposition, we obtain from (1) the two recurrences

$$\begin{aligned} v[j] &= rw[j] - q_{Hj+1}(kd) \\ w[j+1] &= v[j] - q_{Lj+1}d \end{aligned} \tag{2}$$

with quotient digit selection functions

$$\begin{aligned} q_{Hj+1} &= SEL_H(\widehat{rw[j]}, \widehat{d}) \\ q_{Lj+1} &= SEL_L(\widehat{v[j]}, \widehat{d}) \end{aligned}$$

Values for r and k are shown in Table II.

The digit-recurrence algorithm converges if

$$|w[j]| \leq \rho d \tag{3}$$

where ρ is the redundancy factor [12] and is a function of the quotient-digit set and the radix r (see Table II). Therefore, to ensure convergence for the given redundancy, the recurrence is initialized with a scaled value of the dividend such that $w[0] = \text{scaled}(x) < \rho d$.

TABLE II
DIGIT-RECURRENCE PARAMENTERS

	radix-10	radix-16
r	10	16
k	5	4
ρ	7/9	2/3

IV. Combined divider design

A high-level diagram of the combined BID/BFP divider is shown in Fig. 1. The high-level blocks include a normalization block, a recurrence block and a convert-and-round unit. All these blocks are shared to perform BID or BFP division, except the normalization block which is only used in the case of BID-encoded DFP operands. The high-level design is completed by the logic to compute the exponent, the sign ($S_q = S_x \oplus S_d$) and the controller which is partially shown in the figure (counter). The inputs to the unit are: the significands of the dividend, M_x, and the divisor, M_d; the exponents, E_x, and E_d; and signs of both operands, S_x and S_d. The operands may either both be binary64 or both be BID-encoded decimal64 numbers. The inputs to the recurrence are the normalized BID-encoded numbers or the BFP significands, x and d. There is a control CR signal not shown in the diagram that manages the execution of BID or BFP division via multiplexers. The output from the convert-and-round block is the non-normalized BID or normalized BFP quotient, M_q.

The rest of this section describes the implementations of the blocks shown in Fig. 1 in more detail.

A. Normalization

As mentioned before, the operands should be normalized in order to apply the digit-recurrence algorithm. As BFP significands are already normalized, this unit only performs normalization on 64-bit BID-encoded significands. Normalization of DFP operands corresponds to multiplying the operands

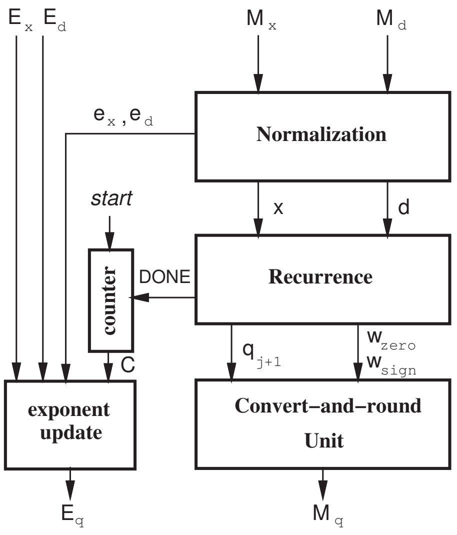

Fig. 1. Architecture of combined divider.

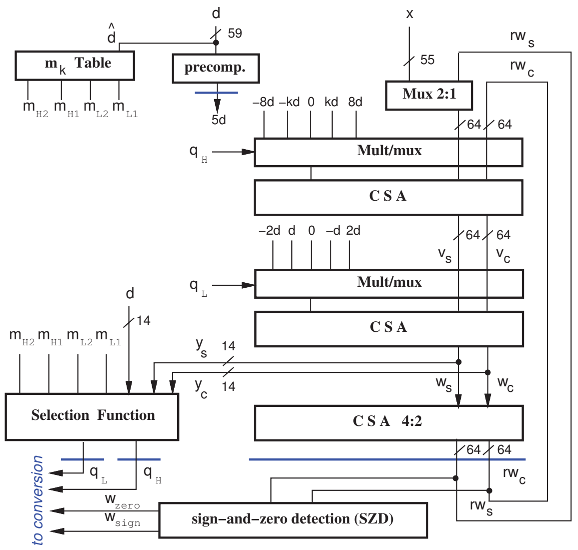

Fig. 3. Recurrence unit.

by the required power of ten. This is performed in two steps: 1) obtain the power of ten required for the normalization; and 2) multiply the operand by the corresponding power of ten using a rectangular multiplier. A detailed diagram of the hardware needed to carry out the normalization is shown in Fig. 2. More details on the normalization unit are found in [11].

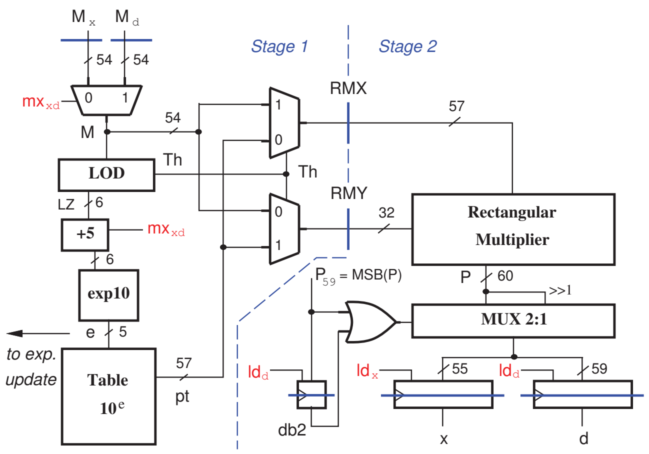

Fig. 2. Normalization unit for BID-encoded significands.

B. Combined Recurrence Unit

Fig. 3 shows the hardware to perform the retimed recurrence

$$\begin{aligned} v[j] &= rw[j-1] - q_{Hj}(kd) \\ w[j] &= v[j] - q_{Lj}d \end{aligned} \qquad (4)$$

Furthermore, to speed-up the iteration time, the residual $w[j]$ (and $v[j]$) is implemented in carry-save format. The position of registers is indicated with a thicker (blue) horizontal line.

The residual is initialized to $w[0]$ with a scaled value of x that for radix-10 is determined in the normalization step and for radix-16 is $x/16$.

The selection constants for radix-10 are the same used in [11], while the selection constants used for radix-16 are listed in Table III. For both radices, the m_k constants are chosen such that $m_{k-1} = -m_{k2}$ and $m_{k0} = -m_{k1}$. In this way, only one constant is stored per $\hat{d}$ interval, and the other is obtained by a two's complement. The combined radix-10 and radix-16 selection function is shown in Fig. 4. The selection of the quotient-digit is done by preloading the corresponding selection constants and by comparing [13] them with the truncated residual $y = \widehat{w}[j]$. Moreover, q_L is calculated speculatively for all possible values of q_H and the correct q_L is selected as soon as q_H is computed (Fig. 4).

The radix-switch CR (not shown in Fig. 3) selects the correct values to be used according to the radix.

radix-10:

- Selection constants: m_{H1}, m_{L2} and m_{L1} for radix-10;
- Precomputed multiple of d: $5d$ (input to q_H Mult/mux);
- Shifted w_s and w_c values to have $10w = 8w + 2w$ in the 4:2 CSA before the rw registers.

radix-16:

- Selection constants: m_{H2}, m_{H1}, m_{L2} and m_{L1} for radix-16;
- Shifted d: $8d$ and $4d$ (input to q_H Mult/mux);
- Shifted w_s and w_c values to have $16w = 8w + 8w$.

The negatives multiples of d are obtained by inverting the bits (one's complement) and by setting the carry-in to one in the CSA. The sign-and-zero detection (SZD) unit determines the sign of the residual (w_{sign}), and when the residual is zero ($w[j] = 0$). In the latter case, the quotient is exact and for decimal division the signal w_{zero} is used to stop the execution of the division if the preferred exponent is obtained. Otherwise, to produce decimal results that comply with IEEE 758-2008, the division should not be stopped until the value with the preferred exponent or closest to the preferred exponent is produced. In case of binary division, if the quotient is exact

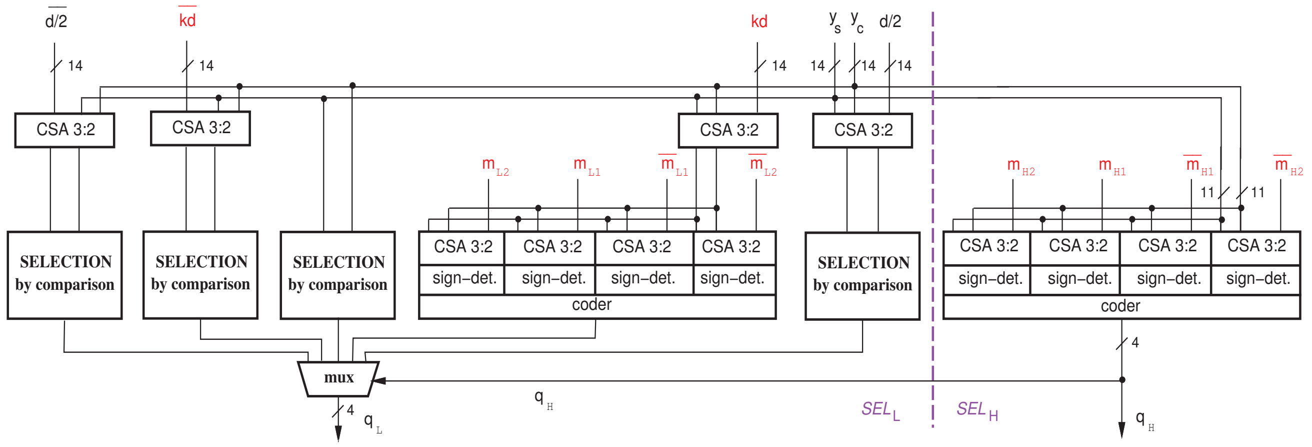
d/2
kd
kd
y_s y_c d/2
14
CSA 3:2
SELECTION by comparison
m_{L2} m_{L1} $\overline{m}_{L1}$ $\overline{m}_{L2}$
m_{H2} m_{H1} $\overline{m}_{H1}$ $\overline{m}_{H2}$
sign–det.
coder
mux
q_H
q_L
SEL_L
SEL_H
11
4

Fig. 4. Selection function.

TABLE III
SELECTION CONSTANTS FOR RADIX-16 DIVISION

$\hat{d}$	m_{H2}	m_{H1}	m_{L2}	m_{L1}
1.000_2	50	16	13	4
1.001_2	56	16	14	4
1.010_2	66	20	16	5
1.011_2	68	20	17	5
1.100_2	72	24	18	6
1.101_2	80	24	20	6
1.110_2	88	28	22	7
1.111_2	88	28	22	7

the division is continued until $1 \leq M_q < 2.0$ (the result significand is normalized). Both w_{sign} and w_{zero} signals are used for rounding as explained in the next section.

C. Combined Convert-and-Round unit

The on-the-fly conversion and rounding implemented in [11] can be easily be adapted to the radix-16 case because the significands of BID-encoded numbers are binary integers. The quotient-digits are converted and assimilated by the unit shown in Fig. 5. At each iteration j, the quotient-digits $q_j = kq_H + q_L$ are converted from signed-digit (SD) to the two's complement digit, B. Then, the partial quotient (Q) is updated as

$$Q[j] \leftarrow rQ[j-1] + B \quad (5)$$

In addition to q_j, $q_j - 1$ and $q_j + 1$ are stored because they are used for rounding. As Figure 5 shows, in radix-10 division, the detection of exact division (w_{zero}=1) occurs at the same time as the q_j conversion and assimilation. Therefore, the assimilation of digit q_j is delayed one cycle.

Rounding is performed in the same unit as conversion and assimilation. In this work, we only consider the rounding mode *roundTiesToEven*. The other rounding modes can be implemented similarly. Rounding is performed in the least-significant digit of the quotient to be assimilated in Q, q_R. The rounding is performed by adding a rounding amount to the least-significant digit to be assimilated, and checking the conditions shown in Table IV.

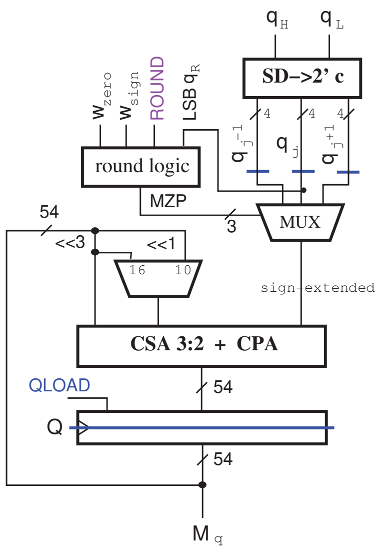

Fig. 5. Convert-and-round unit.

The rounding amount is

$$R = U - w_{sign} - (w_{zero} \text{ AND } \overline{\text{LSB}(q_R)})$$

where U is 5 for radix-10 and 8 or 4 (depending on the scaling of the dividend[1]) for radix-16. Following the conditions shown in Table IV, the last digit to assimilate is q_R, $q_R - 1$ or $q_R + 1$.

V. IMPLEMENTATIONS AND COMPARISONS

In this section, we present the results from evaluating the combined BID/BFP divider. The divider has been modeled in RTL-level VHDL and was simulated using Synopsys VSS and synthesized by using Synopsys Design Compiler and the STM 90nm CMOS standard cell library. The synthesis results are shown in Table V, along with the results of the BID standlone

[1]The first quotient-digit produced is either 1 or 0.

TABLE IV
ROUNDING CONDITIONS FOR RADIX-R

Condition	digit to assimilate
$(B+R) \geq r$	$q_R + 1$
$0 \leq (B+R) < r$	q_R
$(B+R) < 0$	$q_R - 1$

divider presented in [11]. In this table, T_C indicates the critical path delay.

TABLE V
COMPARISON STANDALONE BID AND COMBINED BID/BFP DIVIDER

Timing (ns)	BID*	BID/BFP	
T_C	1.0	1.0	
Latency r-10	24.0	24.0	
Latency r-16	-	17.0	
Area (μm^2)	BID*	BID/BFP	
normalization	83,000	85,000	+2%
recurrence	33,300	51,900	+56%
convert-round	10,600	12,700	+20%
controller	660	680	
divider	127,000	149,000	+17%

* BID divider of [11]

The results indicate that adding support to perform BFP division in a BID divider does not change the clock period of the standalone BID divider. However, the total area is increased by 17%, mostly due to the increased area of the recurrence block. Furthermore, the normalization block for BID operands has a large impact on the total area of each divider. In particular, the rectangular multiplier of the normalization unit has an area of $61,000 \mu m^2$.

The data in Table VI show the number of cycles required by each of the blocks of the combined divider of Fig. 1. We can see that BID normalization penalizes BID division again. The results of the combined BID/DFP divider are compared with the DPD/BFP combined unit of [10] and are shown in Table VII. The results show that the BID/BFP divider can be clocked faster. This was somewhat expected as in the DPD/BFP divider the same datapath is shared by BCD and binary numbers. The total operation latencies are similar because BCD normalization takes fewer cycles. The area of the recurrence plus conversion-and-rounding units of the combined BID/BFP is 20% less than the combined DPD/BFP unit. However, the normalization in the combined BID/BFP divider has a large impact on the area.

TABLE VI
COMBINED BID/BFP DIVISION UNIT DELAY BREAKDOWN

	radix-10	radix-16
Normalization	4	-
Initialization	1	1
Recurrence	17	14
Rounding digit	1	1
Rounding	1	1
TOTAL	24	17

TABLE VII
COMPARISON COMBINED DPD/BFP AND COMBINED BID/BFP DIVIDER

	DPD/BFP*		BID/BFP	
Timing	cycles	ns	cycles	ns
T_C		1.05		1.00
Latency r-10	2+20=22	23.1	4+20=24	24.0
Latency r-16	16	16.8	17	17.0
Area (μm^2)	DPD/BFP		BID/BFP	
normalization	12,500		85,000	+680%
rec. + C.& R.	78,500	+20%	65,000	
divider	91,000		149,000	+74%

* DPD/BFP divider of [10]

VI. CONCLUSIONS

This paper presents a combined decimal and binary floating-point divider for BID-encoded decimal numbers. Relative to a standalone BID divider, the combined BID/BFP divider has the same clock period, the same latency for BID division, and only a 17% increase in area. This indicates that a combined BID/BFP divider may have advantages over separate BID and BFP divider designs. A significant portion of the combined divider's area is due to a large binary multiplier in the normalization unit. However, this multiplier or a similar binary multiplier could also be used to help perform other BID operations, such as addition, subtraction, and multiplication.

REFERENCES

[1] M. F. Cowlishaw, "Decimal Floating-Point: Algorism for Computers," in *IEEE Symposium on Computer Arithmetic*, 2003, pp. 104–111.

[2] IEEE Computer Society Std., "IEEE Standard for Floating-Point Arithmetic," August 2008.

[3] L.-K. Wang and M. Schulte, "A Decimal Floating-Point Divider Using Newton–Raphson Iteration," *J. VLSI Signal Process. Syst.*, vol. 49, no. 1, pp. 3–18, 2007.

[4] A.Vazquez, E.Antelo, and P.Montuschi, "A Radix-10 SRT Divider Based on Alternative BCD Codings," in *IEEE International Conference on Computer Design (ICCD)*, Lake Tahoe, USA, October 2007, pp. 280–287.

[5] T. Lang and A. Nannarelli, "A Radix-10 Digit-Recurrence Division Unit: Algorithm and Architecture," *IEEE Trans. Computers*, vol. 56, no. 6, pp. 727–739, 2007.

[6] C. Tsen, S. Gonzalez-Navarro, M. J. Schulte, B. J. Hickmann, and K. Compton, "A Combined Decimal and Binary Floating-Point Multiplier," in *IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP)*, 2009, pp. 8–15.

[7] A.Vazquez and E.Antelo, "Conditional Speculative Decimal Addition," in *7th Conference on Real Numbers and Computers (RNC7)*, Nancy, France, 2006, pp. 47–57.

[8] A.Vazquez, E.Antelo, and P.Montuschi, "A New Family of High-Performance Parallel Decimal Multipliers," in *IEEE Symposium on Computer Arithmetic*, Montpellier, France, June 2007, pp. 195–204.

[9] B. J. Hickmann, M. A. Erle, and M. J. Schulte, "Improved Combined Binary/Decimal Fixed-Point Multipliers," in *IEEE International Conference on Computer Design (ICCD)*, Lake Tahoe, CA, October 2008, pp. 87–94.

[10] T. Lang and A. Nannarelli, "Combined Radix-10 and Radix-16 Division Unit," in *Proc. of 41st Asilomar Conference on Signals, Systems, and Computers*, Nov 2007.

[11] T. Lang and A. Nannarelli, "Division Unit for Binary Integer Decimals," in *IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP)*, 2009, pp. 1–7.

[12] M. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic, 1994.

[13] N. Burgess and C. Hinds, "Design Issues in Radix-4 SRT Square Root and Divide Unit," in *35th Asilomar Conf. Signals, Systems, and Computers*, 2001, pp. 1646–1650.