2009 19th IEEE International Symposium on Computer Arithmetic

A High-Performance Significand BCD Adder with IEEE 754-2008 Decimal Rounding*

Alvaro Vázquez and Elisardo Antelo
Department of Electronic and Computer Engineering
University of Santiago de Compostela, SPAIN
alvaro.vazquez@usc.es, elisardo.antelo@usc.es

Abstract

We present a new method and architecture to merge efficiently IEEE 754-2008 decimal rounding with significand BCD addition and subtraction. This is a key component to improve several decimal floating-point operations such as addition, multiplication and fused multiply-add. The decimal rounding unit is based on a direct implementation of the IEEE 754-2008 rounding modes. We show that the resultant implementations for IEEE 754-2008 Decimal64 (16 precision digits) and Decimal128 (34 precision digits) formats reduce significantly the area and latency required for significand BCD addition/subtraction and decimal rounding in previous high-performance decimal floating-point adders.

1. Introduction

The revision of the IEEE 754-1985 and IEEE 854-1987 floating-point standards (IEEE 754-2008) [4], incorporates specifications for decimal arithmetic [2]. In this way, the first IEEE 754-2008 compliant DFUs (decimal floating-point units) are already in the market [3, 6]. These units present low performance since design efforts are focused on providing support for the Decimal128 format (34 precision digits) in a reduced area. On the other hand, recent implementations of DFP (decimal floating-point) adders from academical research [9, 12] proposed fully parallel architectures for high performance.

In this paper, we concentrate on the addition/subtraction of the BCD coefficients (significand BCD addition) and decimal rounding due to their significant contribution to the total delay of DFP addition [9]. The previous high-performance DFP adders use different schemes to implement these two operations: Thompson, Karra and Schulte [9] perform decimal rounding after significand BCD addition. The significand BCD addition consists of a pre-correction of the BCD input operands (performed in constant time), a binary carry-propagate compound addition (logarithmic delay) and a decimal post-correction (constant-time operation, but more complex than the pre-correction). In this case, decimal rounding requires a conditional +1 ulp (unit in the last place) increment of the significand BCD sum (logarithmic delay). On the other hand, Wang and Schulte [12] combine significand BCD addition with rounding by injection [7] to reduce the latency of the previous scheme. To avoid another carry-propagate addition, required for correct rounding when the most significant digit of the injected BCD sum is not zero, they need to determine the number of trailing 9's of the BCD sum. This evaluation (logarithmic delay) is performed after the binary carry-propagation, in parallel with the decimal post-correction.

We adapt a technique proposed for 10's complement BCD addition [11] to implement sign-magnitude BCD addition/subtraction using a binary compound adder and simple and fast decimal pre-correction and post-correction schemes. Moreover, the method proposed in this paper does not require any additional carry propagation or trailing 9's detection for decimal rounding. The modifications introduced into the sign-magnitude BCD adder to support decimal rounding only increment its critical path delay by a small constant amount, independently of the number of input digits.

The paper is organized as follows. Section 2 contains some important issues about DFP addition and describes a recent high-performance IEEE 754-2008 DFP adder [12]. This implementation has the best performance reported to date. In Section 3 we introduce the new algorithm for combined significand BCD addition and decimal rounding. In Section 4 we detail the proposed ar-

*This work was supported in part by the Ministry of Education and Science of Spain under contract TIN 2007-67537-C03. The authors would also like to thank IBM for their support.

1063-6889/09 $25.00 © 2009 IEEE
DOI 10.1109/ARITH.2009.30

chitecture. Area and delay evaluation results are shown in Section 5. We use a model based on logical effort [8] to provide a quantitative delay analysis. We compare the proposed architecture with the significand BCD adder and rounding unit implemented in [12]. Finally, the conclusions are summarized in Section 6.

2. Overview of IEEE 754-2008 DFP addition

We consider the high-performance DFP adder proposed in [12]. This architecture was originally proposed for the IEEE 754-2008 Decimal64 format but can be easily extended for the Decimal128 format. A DFP addition/subtraction is carried out as follows. Operands FX and FY, stored in DPD (densely packed decimal) format, are unpacked in a sign bit (s_X,s_Y), a biased binary integer exponent (E_X, E_Y) and a p-digit ($p = \{16, 34\}$) BCD non-normalized coefficient or significand (X, Y).

After unpacking the DPD operands, the exponents are compared using a binary sign-magnitude adder of few bits (10 bits for the Decimal64 format). This unit computes the exponent difference ($E_D = |E_X - E_Y|$) and $sign(E_X - E_Y)$. In parallel, the number of leading zeroes (lz_X, lz_Y) of both input coefficients are determined using two leading zero detectors (LZD). The operands are swapped if $E_X < E_Y$ ($sign(E_X - E_Y) = 1$). Next, the BCD coefficients UF (with higher exponent $E_{UF} = max(E_X, E_Y)$) and LF (with lower exponent $E_{LF} = min(E_X, E_Y)$) are aligned, so the resulting coefficients U and L have the same exponent. The number of leading zeroes of UF (lz_{UF}) is determined speculatively from lz_X or lz_Y. Two cases may occur:

1. $E_D \leq lz_{UF}$**.** The coefficient UF is shifted left E_D digits and then added to the other unshifted coefficient ($L = LF$).
2. $E_D > lz_{UF}$**.** Both operands UF and LF are shifted. The operand with the larger exponent (UF) is shifted left lz_{UF} digits and the operand with the smaller exponent (LF) is shifted $E_D - lz_{UF}$ digits right.

The variable eop defines the effective operation ($eop = 0$ effective addition and $eop = 1$ for effective subtraction). In case of effective addition, UF and LF are shifted one extra digit to the right to simplify rounding operations. The effective operation is determined as $eop = (-1)^{s_U + s_L + d_s}$, where s_U, s_L are the signs of input operands after operand swapping and d_s indicates the operation specified by the instruction ($d_s = 0$ for addition, $d_s = 1$ for subtraction). Thus, the left and right shift amounts for operand alignment are obtained faster as $lsa = min(E_D, lz_{UF}) + eop$ and $rsa = min\big(max(E_D - lz_{UF}, 0) + \overline{eop}, p+3\big)$. The alignment is performed using two barrel shifters, which perform decimal shifts.

The addition/subtraction of the aligned BCD coefficients (significand BCD addition) results in a BCD magnitude $|R| = |U + (-1)^{eop} L|$ and a sign bit $sign(R)$. High-performance implementations use a BCD compound adder to compute $|R|$. Thus, $|R| = S$ ($S = U + L$) if $eop = 0$, $|R| = S + 1$ ulp ($S = U + \neg L$) if $eop = 1$ and $U \geq L$, and $|R| = \neg S$ ($S = U + \neg L$) if $eop = 1$ and $U < L$. The symbol $\neg$ represents the 9's complement and ulp is a unit in the last (least significant) place.

To get the correctly rounded result, significand BCD addition is carried out with three extra precision digits (guard digit GD, round digit RD, and sticky bit st), so $|R| = \sum_{i=-3}^{p-1} |R|_i \ 10^i$. The sticky bit L_{st} is obtained by OR-ing the bits of LX shifted out to the right of the RD position. The $p + 1$-digit rounded coefficient RX (it includes the guard digit) is given by (MSD means the Most Significant Digit)

$$RX = \begin{cases} Round(|R|)_{p+1} & \texttt{If } |R|_{MSD} = 0 \\ Round(|R|)_p & \texttt{Else} \end{cases} \quad (1)$$

where $Round(|R|)_p$ represents $|R|$ rounded to p precision digits as stated for the corresponding IEEE 754-2008 decimal rounding mode [4]. RX is shifted one digit to the left when $RX_{MSD} = 0$ so the leading zero is discarded. Finally, the p-digit rounded coefficient RF is obtained from the most p significant digits of the resultant RX. In the next Section we describe our method for combined significand BCD addition and rounding, that is, to compute RX. The scheme for significand BCD addition and decimal rounding proposed by Wang and Schulte [12] is analyzed and compared with our proposal in Section 5.2.

The biased exponent of the result E_{RF} is computed as $E_U - min(E_D, lz_U)$ and incremented by 1 unit if RX_{MSD} is not zero. The $sign(RF)$ is computed as $sign(RF) = \overline{eop}\ s_U \vee eop\ sign(R)$ (we represent the logical OR as $\vee$ and the logical AND by a blank space between variables). Finally, $sign(RF)$, E_{RF} and RF are packed to DPD. The post-processing unit handles special cases.

3. Method for Significand BCD Addition with Rounding

3.1. General Description

Before introducing the algorithm for the computation of the rounded BCD significand RX, given by Expression

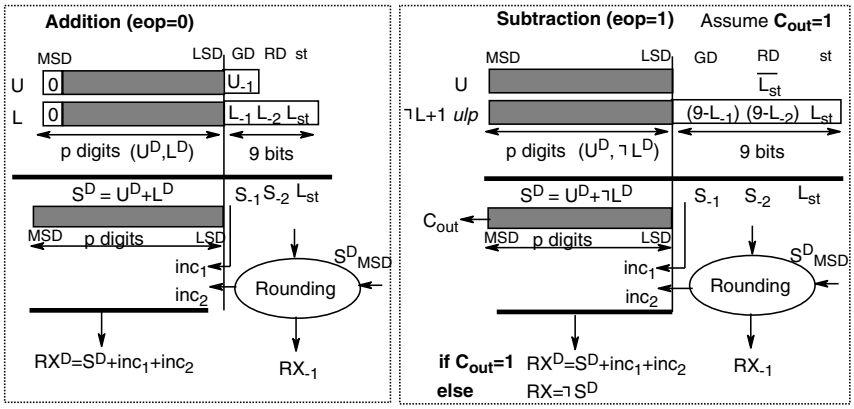

Figure 1. Case description of significand BCD addition and rounding.

(1), we present in Fig. 1 a description for both addition ($eop = 0$) and subtraction ($eop = 1$) when combined with rounding. The aligned BCD input operands U ($p+1$ digits) and L ($p+3$ digits) are of the form

$$U = \sum_{i=-1}^{p-1} U_i \, 10^i \;\; , \;\; L = \sum_{i=-3}^{p-1} L_i \, 10^i$$

Digits U_0, L_0 are placed at the LSD (least significant digit) position, U_{-1} and L_{-1} are at the GD (guard digit) position, L_{-2} is at the RD (round digit) position and $L_{-3} = L_{st} \in \{0, 1\}$ is the sticky bit. For effective addition, the MSDs of both operands are zero, that is, $U_{p-1} = L_{p-1} = 0$. For effective subtraction, $U_{-1} = 0$.

To incorporate rounding into the significand BCD addition, we perform the computation in two parts. We split BCD operands U and L in a p-digit most significant part (U^D, L^D) and a least significant one, which includes the digits/bits placed at the guard (U_{-1}, L_{-1}), round (L_{-2}) and sticky (L_{st}) positions. The p-digit operands S^D ($\neg S^D$), $S^D + 1$ and $S^D + 2$ with:

$$S^D = \begin{cases} U^D + L^D & \texttt{If } eop = 0 \\ U^D + \neg L^D & \texttt{Else} \end{cases}$$

are computed speculatively. Two signals, inc_1 and inc_2 represent the carries into the LSD of S^D due to the addition of the least significant part and the rounding.

Signal inc_1 is a decimal carry into the LSD of S^D due to the addition of the 3 least significant digits of U and L. For effective addition (left side in Fig. 1), inc_1 results from the 1-digit BCD sum $U_{-1} + L_{-1}$. For effective subtraction (right side in Fig. 1), the 9's complement of L is computed first. Moreover, when the carry-out C_{out} of S^D is one (that is, $U - L \geq 0$), a +1 ulp must be added to $\neg L$ to obtain the ten's complement of L. For $C_{out} = 0$ (that is, $U - L < 0$), we only need to compute $RX = \neg(U + \neg L + 1) + 1 = \neg(S^D + 1) + 1 = \neg S^D$ ($LX = L$ is unshifted, $S = S^D$ and no rounding is required). To avoid a dependence on C_{out}, we compute inc_1 speculatively assuming $C_{out} = 1$, and define a variable $cmp = eop\ \overline{C_{out}}$ to obtain later the correct result for the case $U - L < 0$. Therefore, inc_1 is computed as follows

$$inc_1 = \begin{cases} \lfloor (U_{-1} + L_{-1})/10 \rfloor & \texttt{If} (eop = 0) \\ \lfloor (\neg(L_{-1}\, 10 + L_{-2}) + \overline{L_{lsb}})/100 \rfloor & \texttt{Else} \end{cases}$$

Decimal rounding may occur either at the RD or the GD, depending on the MSD of $S^D + inc_1$ (see Section 4.3 for more detail). Rounding generates another decimal carry inc_2 into the LSD of S^D. However, we do not evaluate $S^D + inc_1$ directly, but S^D and $S^D + 1$. Thus, the MSD of $S^D + inc_1$ must be determined from the MSDs of S^D and $S^D + 1$ and from inc_1. Moreover, the $p + 1$-digit rounded BCD significand results from $RX = RX^D \& RX_{-1}$ (we use & to indicate a concatenation), where

$$RX^D = \begin{cases} S^D + T & \texttt{If } cmp = 0 \\ \neg S^D & \texttt{If } cmp = 1 \end{cases} \qquad (2)$$

$T = inc_1 + inc_2$ ($T \in \{0, 1, 2\}$) and RX_{-1} is the digit at GD position obtained after round-off (see Section 4.3). In the next Section we propose an algorithm for combined significand BCD addition and decimal rounding, which uses a single binary compound adder to compute speculatively the values S^D ($\neg S^D$), $S^D + 1$ and $S^D + 2$.

3.2. Algorithm

To obtain the three related sums S^D (or $\neg S^D$), $S^D + 1$ and $S^D + 2$ using the same compound adder, we split the

evaluation of S^D as $S^D = S^H \& S^D_{lsb}$, where S^D_{lsb} is the lsb (least significant bit) of S^D and S^H is a p-digit BCD operand with a lsb of zero. At the lsb position of S^D we should add the lsbs of U^D_0 and L^D_0, inc_1 and inc_2, resulting in two carries to the part that computes S^H. To simplify the implementation, as it is done for binary floating-point adders [1, 7], one of the carries (produced by the addition of the lsbs of U^D_0 and L^D_0) is propagated using a carry-save addition, resulting in a single bit S^D_{lsb} at the lsb position. The other carry, given by

$$inc = \lfloor (inc_1 + inc_2 + S^D_{lsb})/2 \rfloor$$

produces an increment of $+2$ inc ulps in S^H. To have an efficient implementation we need to compute speculatively the BCD sums S^H and $SI^H = S^H + 2$, selecting the suitable value according to inc. Moreover, when $cmp = 1$ we just compute $\neg S^H$.

In the following we explain how the carry-save addition and the subsequent carry-propagate addition to compute S^H and SI^H are performed. It is important to note that to perform the decimal carry-propagate addition so that the decimal carries correspond to binary carries at the same position (to use a binary carry tree), a $+6$ have to be added to each digit position of the BCD operands [5]. A BCD digit of the result is correct if the carry out from that digit is one (addition of digits plus previous carry greater than 9). However, if the carry out is zero, a subtraction of 6 to that digit should be performed.

Carry-save addition: apart from using this step to propagate one of the carries, we also add the $+6$ in each digit position to simplify the carry-propagate addition. Therefore the computation performed at digit i is: $U_i + L_i + 6$ for effective addition, and $U^D_i + \neg L^D_i + 6 = U_i + (15 - L_i) = U^D_i + \overline{L^D_i}$ for effective subtraction. To have an efficient implementation, we use a binary carry-save adder that produces a p-digit carry operand Opc (with digits $Opc_i \in [0, 9]$) and a p-digit sum operand $Ops = Ops^H \& S^D_{lsb}$ (with digits $Ops_i \in [0, 15]$) such that $U^D + L^D + \sum_{i=0}^{p-1} 6\ 10^i = Ops + 2 \times Opc$. However, instead of computing $2 \times Opc$, we perform a 1-bit left shift of the carry operand Opc ($L1_{shift}[Opc]$). The decimal digit i of $L1_{shift}[Opc]$ is equal to the digit i of $2 \times Opc$ when the bit shifted out to digit $i + 1$ is zero (when the sum of the input digits is lower than 16). Otherwise the resultant digit is $(L1_{shift}[Opc])_i = (2 \times Opc)_i - 6$ (when the sum of the input digits is greater or equal 16) since the one bit left shift adds a 16 to digit $i + 1$ instead of 10.

Carry-propagate compound addition: this step produces the BCD sums S^H ($\neg S^H$) and $S^H + 2$ from operands $L1_{shift}[Opc]$ and Ops^H (both have a lsb equal to 0). Therefore, apart from the decimal carries generated due to the addition of the input operands of the compound adder, we need to detect also a carry propagation due the plus 2. To detect both carry propagations using a binary carry propagate tree, the sum of the input digits at each position should be in excess to 6. We have the following cases:

- Range of $U^D_i + L^D_i$ or $U^D_i + \neg L^D_i$ (depending on the effective operation) in $[16, 18]$: the sum of input digits i to the adder after the carry-save step is not in excess to 6. Therefore an additional $+6$ digit addition must be performed (required to detect a carry propagation due the plus one). The resultant sum digit is in BCD excess to 6.

- Range of $U^D_i + L^D_i$ (or $U^D_i + \neg L^D_i$) in $[10, 15]$: the sum of input digits i to the adder is already in excess to 6. However the output digit is in BCD, because the decimal carry out to next digit, C_{i+1}, is one.

- Range of $U^D_i + L^D_i$ (or $U^D_i + \neg L^D_i$) in $[0, 8]$: both, the input and the output sum digits are in BCD excess to 6.

- $U^D_i + L^D_i$ (or $U^D_i + \neg L^D_i$) equal to 9: This case depends on the decimal carry input C_i, produced either in the carry-save adder ($C1_i$) or in the carry-propagate addition ($C2_i$). If the carry input ($C_i = C1_i \vee C2_i$) is 0 then, both the input and output sum digits are in BCD excess to 6. If the carry input C_i is 1, then the output sum digit is in BCD.

We obtain the BCD sums S^H ($\neg S^H$) and $S^H + 2$ by means of a binary compound addition and a subsequent decimal correction (digitwise) of the binary sums. To have an efficient implementation of the decimal post-correction, we need all the output digits in the same excess. In this case, the decimal post-correction is independent of the decimal carries generated in the carry-save and carry-propagate adders.

Given the above cases, the only alternative is to have all the output digits in excess to 6. Therefore, we also have to add an additional 6 when $U^D_i + L^D_i$ (or $U^D_i + \neg L^D_i$) are in the range $[10, 15]$. To detect both ranges $[10, 15]$ and $[16, 18]$ we use the decimal generate function G_i, defined by

$$G_i = \begin{cases} 1 & \texttt{If } U^D_i + L^D_i \geq 10 \texttt{ and } eop = 0 \\ 1 & \texttt{If } U^D_i + \overline{L^D_i} \geq 16 \texttt{ and } eop = 1 \\ 0 & \texttt{Else} \end{cases} \quad (3)$$

We add this additional 6 to the digits of operand $L1_{shift}[Opc]$ conditionally, obtaining the following p-

[Algorithm: Significand BCD Addition and IEEE 754-2008 Decimal Rounding]

Inputs: $U = U^D + U_{-1}\,10^{-1}, L = L^D + \sum_{i=-3}^{-1} L_i\,10^i, eop$

Output: $RX = \sum_{i=-1}^{p-1} RX_i\,10^i$

In parallel, compute A and B:

A. Computation of the BCD sums S^H and $S^H + 2$.

A.1. Pre-correction.

Inputs: U^D, L^D, eop

For (i=0;i<p;i++){

- A.1.1) Add +6 $L_i^* = \begin{cases} L_i^D + 6 & \texttt{If } eop = 0 \\ (\neg L_i^D) + 6 = \overline{L_i^D} & \texttt{Else} \end{cases}$
- A.1.2) Binary carry-save addition (Ops_i, Opc_i) obtained from $U_i^D + L_i^{D*}$
- A.1.3) Compute G_i defined in (3)
- A.1.4) Add +3 (Conditional) $Opd_i = Opc_i + 3\,G_i$

}

Outputs: $C1_{out} = Opc_{p-1,3}, S_{lsb}^D = Ops_{0,0}, Ops^H = Ops - Ops_{0,0}, Opd = L1_{shift}[\sum_{i=0}^{p-1} Opd_i\,10^i]$

A.2. Binary compound addition:

Inputs: $Ops^H, Opd, eop, C2_0 = 0, GP_0 = 1$

For (i=0;i<p;i++){

- A.2.1) Binary carry-propagate computation:

$$C2_{i+1} = \lfloor (Ops_i^H + Opd_i + C2_i)/16 \rfloor$$
$$GP_{i+1} = \lfloor (Ops_i^H + Opd_i + 1)/16 \rfloor\, GP_i$$

- A.2.2) Decimal digit addition:

$$S_i^* = \begin{cases} 6 & \texttt{If}\big(U_i^D + L_i^* == 15 \texttt{ AND } (C1_i \texttt{ OR } C2_i) == 1\big) \\ \mathrm{mod}_{16}(Ops_i^H + Opd_i + C2_i) & \texttt{Else} \end{cases}$$

$$SI_i^* = \begin{cases} 6 & \texttt{If}\big(U_i^D + L_i^* == 15 \texttt{ AND } (C1_i \texttt{ OR } C2_i\, GP_i) == 1\big) \\ \mathrm{mod}_{16}(Ops_i^H + Opd_i + C2_i\, GP_i) & \texttt{Else} \end{cases}$$

}

Outputs: $S^*, SI^*, C2_{out} = C_p$

A.3. Post-correction:

Inputs: S^*, SI^*

For (i=0;i<p;i++){ $S_i^H = S_i^* - 6, SI_i^H = SI^* - 6, \neg S_i^H = \overline{S_i^*}$ }

Outputs: $S^H, SI^H, \neg S^H$

B. Decimal Rounding:

Inputs: $U^{-1}, L^{-1}, L^{-2}, L^{st}, eop, S_{lsb}^D, SI_{MSD}^*, S_{MSD}^*$

Outputs: inc_1 and inc_2, RX_{-1} obtained from the decimal rounding rules (Section 4.3)

C. Selection:

$inc = \lfloor (inc_1 + inc_2 + S_{lsb}^D)/2 \rfloor, R_{lsb} = \mathrm{mod}_2(inc_1 + inc_2 + S_{lsb}^D)$

$$RX^D = \begin{cases} (S^H, R_{lsb}) & \texttt{If}\big(inc < 2 \texttt{ AND } cmp = 0\big) \\ (S^H, R_{lsb}) & \texttt{If}\big(inc \geq 2 \texttt{ AND } cmp = 0\big) \\ (\neg S^H, \overline{S_{lsb}^D}) & \texttt{If}(cmp = 1) \end{cases}$$

Outputs: inc_1, inc_2, RX^D

Figure 2. Proposed algorithm.

digit operand:

$$Opd = L1_{shift}[\sum_{i=0}^{p-1}(Opc_i + 3\,G_i)\,10^i] \qquad (4)$$

This addition is a digitwise operation (does not generate a carry-propagation between digits) since $Opc_i \in [0, 9]$ and $Opc_i + 3\,G_i \in [0, 12]$. Moreover, we have to add the additional +6 in the case of $U_i^D + L_i^D$ (or $U_i^D + \neg L_i^D$) equal to 9 and the decimal carry input C_i equal to 1. In this case, the output sum digit i is equal to 0 (BCD no excess), but we need it to be in excess to 6. This is performed introducing a slight modification in the xor sum cells of the binary adder, out of its critical path. We present this modification in Section 4.2.

Therefore, the modified binary compound adder computes the $(4p-1)$-bit sums $S^* = Ops^H + Opd$ and $SI^* = S^* + 2$ (note that the lsbs of Ops^H and Opd are zeroes). We obtain the BCD sums S^H and $S^H + 2$ by subtracting 6 (digitwise) from each digit (in excess to 6) of binary sums S^* and SI^* respectively. On the other

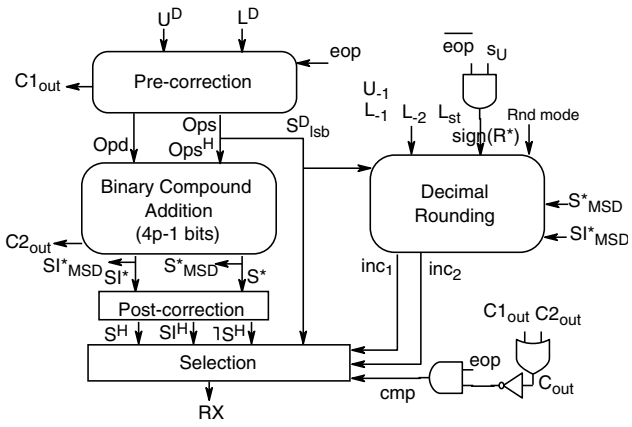

Figure 3. High-level implementation.

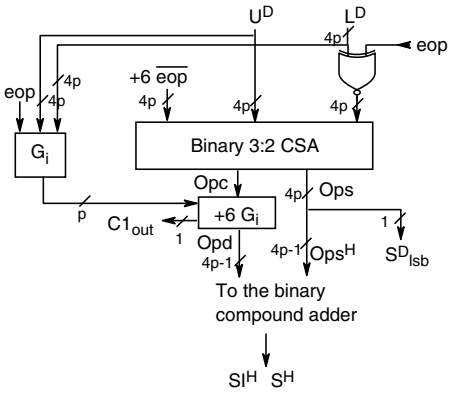

Figure 4. Pre-correction stage.

hand, the 9's complement of S^H is obtaining inverting the bits of S^*, that is, $\neg S^H = \overline{S^*}$. Finally, the digits of RX^D are selected from S^H ($\neg(S^H)$) and $S^H + 2$ depending on the values of cmp and inc as stated in Expression (2). The signal cmp is determined from the carry-outs ($C1_{out}$, $C2_{out}$) of the binary carry-save and binary carry-propagate adders as

$$cmp = eop\ \overline{C_{out}} = eop\ \overline{C1_{out} \vee C2_{out}}$$

The proposed algorithm for the whole significand BCD addition/subtraction with decimal rounding is summarized in Fig. 2. Note that the carry-save addition step described previously corresponds to the pre-correction steps A.1.1 and A.1.2 of Fig 2, while the compound addition step corresponds to steps A.1.3, A.1.4, A.2 and A.3 of Fig. 2.

4. Detailed description of the architecture

In this Section we detail the proposed architecture. A high-level block diagram is shown in Fig. 3. It consists of five stages, which implement the functionality of the algorithm of Fig. 2 : pre-correction, binary compound addition, post-correction, rounding and selection. The functionality and implementation of each block is presented next.

4.1. Pre-correction

To allow the use of binary operations, p-digit BCD operands U^D and L^D are processed in the pre-correction stage as detailed in Fig. 4.

For effective addition ($eop = 0$), we perform the digit additions $U_i + L_i + 6$ in a $4p$-bit binary 3:2 CSA, obtaining the $4p$-bit sum and carry operands Ops and Opc. Each +6 bias, coded in BCD as $(0, \overline{eop}, \overline{eop}, 0)$, is connected to an input of a 4-bit binary 3:2 CSA. The p-digit BCD operands U^D and L^D are introduced into the other two inputs. For effective subtraction ($eop = 1$), the +6 digit additions are performed together with the 9's complement of L by inverting the bits of L_i, that is, $U_i + \neg L_i + 6 = U_i + \overline{L_i}$. Next, the p-digit BCD operands U^D and $\overline{L^D}$ are introduced in the binary 3:2 CSA. The evaluation of $\overline{L^D}\ eop \vee L^D\ \overline{eop}$ is performed in a level of $4p$ XOR gates, which is connected to the fast input of the 3:2 binary CSA.

The decimal carry generates G_i are evaluated in parallel with Opc to reduce the critical path delay. The block G_i implements equations (3) in 4 logic levels. Despite the logic depth of both parallel paths is very similar (4 logic level vs. 2 XOR gate levels), the computation of the G_i is slightly slower than the computation of Opc and Ops due to the higher loads. A quantitative delay analysis is presented in Section 5.1. Finally, the $+6\ G_i$ block (two logic levels) evaluates Opd as indicated by Equation (4).

4.2. Binary compound addition

We may use any conventional binary compound adder, such as a flagged prefix adder [1], to compute the sums $S^* = Ops^H + Opd$ and $SI^* = S^* + 2$. The binary carry recurrence $c_{k+1} = g_k \vee a_k\ c_k$ is implemented in a prefix carry tree of $\log_2(4p-1)$ levels, where g_k and a_k are the binary carry generate and carry OR-propagate functions. The decimal carries $C2_i$ are the binary carries c_k at decimal positions (1 out of 4).

The 4-bit binary sum $Ops_i^H + Opd_i + C2_i$ represents the BCD excess-6 sum digit S_i^* except for $U_i^D + L_i^D = 9$ and $C_i = C1_i \vee C2_i = 1$ (see Section 3.2), which corresponds to the values ($Ops_i^H + Opd_i = 0$, $C2_i = 0$) when $C1_i = 1$, or to ($Ops_i^H + Opd_i = 15$, $C2_i = 1$)

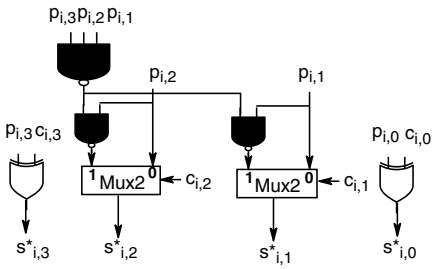

Figure 5. Modified 4-bit binary sum cell.

when $C1_i = 0$. Both cases are detected checking the following boolean expression $p_{i,3}\ p_{i,2}\ p_{i,1}\ c_{i,1} = 1$, where the $p_{i,j} = p_{k-1}$ with $k = 4i + j$, are the binary XOR carry propagates and $c_{i,1} = c_{k-1}$. Then, the sum value 0 ('0000') is transformed into the corresponding excess 6 value $S_i^* = 6$ ('0110'). To perform this detection and correction, we modify each 4-bit binary sum cell as shown in Fig. 5. The additional gates introduced (in black) are out of the critical path (the carry path), so they do not increment the latency of the binary adder.

On the other hand, the length of the decimal carry propagation due to a late +2 ulp increment of the BCD sum S^H is determined by a trailing digit chain of $(k-1)$ 9's and an 8 (the lsb of S^H is zero). This is equivalent to detect the length of the trailing chain of '1' in the sum S^*. A carry due a +2 ulp increment of S^* is propagated to bit k if the binary carry OR-propagate group $a_{k-1:1} = a_{k-1}\ a_{k-2} \ldots\ a_1$ is true. The carries c_k and the carry OR-propagate groups $a_{k-1:1}$ are computed in the same prefix tree [1]. The binary carries of SI^* are obtained as $lc_k = c_k \vee a_{k-1:1}$. The digits SI_i^* are obtained from the binary XOR-propagates $p_{i,j}$ and the binary carries $lc_{i,j}$ $(k = 4i + j)$ using a 4-bit sum cell similar as the one shown in Fig. 5.

4.3. Decimal rounding

In parallel with the binary sum, the decimal rounding unit computes the increment signals inc_1 and inc_2 and the guard digit of the result RX_{-1}. Apart from the rounding mode, the rounding decision depends on the effective operation eop, the guard digits U_{-1}, L_{-1}, the round digit L_{-2}, the sticky bit L_{st}, the MSD and the carry-out C_{out} (always zero for effective addition) of S^* and the MSD of SI^*. Some rounding modes also require $S_{lsb}^D \oplus inc_1$ (nearest even), or the expected sign of the result $sign(R^*) = \overline{eop}\ s_U$ (towards $\pm\infty$). The implementation of the decimal rounding stage is shown in Fig. 6. It uses combinational logic to implement directly a rounding condition for each decimal rounding

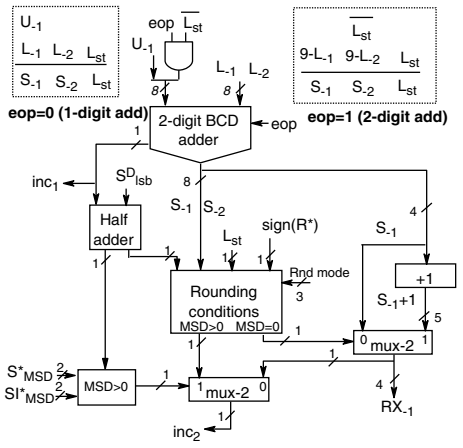

Figure 6. Diagram of decimal rounding.

mode. Moreover, different conditions must be applied when the MSD of the significand result (before rounding) is zero or not zero. The decimal rounding unit can be also implemented using an injection scheme [10].

To simplify this logic, the guard and round digits of U and L are previously assimilated in a 2-digit BCD adder, producing two sum digits S_{-1}, S_{-2} and the decimal carry-out inc_1. The layout of the input operands differs if the effective operation is addition ($eop = 0$, top left corner of Fig. 6) or subtraction ($eop = 1$, top right corner of Fig. 6). For $eop = 0$, inc_1 and S_{-1} are the result of the 1-digit BCD addition $U_{-1} + L_{-1}$, while $S_{-2} = L_{-2}$. For $eop = 1$, the addition of $\neg(L_{-1}\ 10 + L_{-2})$ and the bit $\overline{L_{st}}$ produces two BCD sum digits S_{-1}, S_{-2}, while the decimal carry-out inc_1 is 1 only if $L_{-1} = L_{-2} = L_{st} = 0$. Therefore, the logic of the 2-digit BCD adder can be simplified in order to obtain inc_1 faster.

The second decimal carry out, inc_2, is produced by the rounding. In addition to the five IEEE 754-2008 decimal rounding modes [4], we implement two additional rounding modes [9, 12]: round to nearest down and away from zero. The conditions for each decimal rounding mode are summarized in Table 1, where $s_{GD,0}$ is the lsb of S_{-1}. Depending on the MSD of the unrounded result, the roundoff may produce an increment of +1 unit in S_{LSD} or S_{-1}. The first case ($MSD > 0$) corresponds with a +1 ulp increment of S (inc_2). In the second case ($MSD = 0$), inc_2 is true if the corresponding rounding condition is verified and $S_{-1} = 9$. To obtain RX_{-1} and inc_2 faster, we compute $S_{-1}+1$ (module 10) in parallel with the rounding conditions, and then S_{-1} or $S_{-1}+1$ is selected from the corresponding rounding condition for $MSD = 0$. The condition $MSD > 0$ occurs when $S_{MSD}^H > 0$ or $S^H = 099 \ldots 98$ and $S_{lsb}^D + inc_1 = 2$.

Rounding mode	$MSD > 0$	Rounding condition
To nearest even	0	$(S_{-2} > 5) \vee ((S_{-2} = 5)\ (s_{GD,0} = 1 \vee L_{st} = 1))$
	1	$S_{-1} > 5 \vee ((S_{-1} = 5)\ ((S^D_{lsb} \oplus inc_1) = 1 \vee (S_{-2} > 0) \vee L_{st} = 1))$
To nearest up	0	$S_{-2} \geq 5$
	1	$S_{-1} \geq 5$
To nearest down	0	$S_{-2} > 5 \vee (S_{-2} = 5\ (L_{st} = 1))$
	1	$S_{-1} > 5 \vee (S_{-1} = 5\ (S_{-2} > 0 \vee (L_{st} = 1)))$
Toward $+\infty$	0	$(sign(R^*) = 0)\ (S_{-2} > 0 \vee (S_{-2} = 0\ L_{st} = 1))$
	1	$(sign(R^*) = 0)\ (S_{-1} > 0 \vee (S_{-1} = 0\ (S_{-2} > 0 \vee L_{st} = 1)))$
Toward $-\infty$	0	$(sign(R^*) = 1)\ (S_{-2} > 0 \vee (S_{-2} = 0\ L_{st} = 1))$
	1	$(sign(R^*) = 1)\ (S_{-1} > 0 \vee (S_{-1} = 0\ (S_{-2} > 0 \vee L_{st} = 1)))$
Toward zero	$\{0, 1\}$	0
Away from zero	0	$(S_{-2} > 0)\ \vee (S_{-2} = 0\ (L_{st} = 1))$
	1	$(S_{-1} > 0)\ \vee (S_{-1} = 0\ (S_{-2} > 0 \vee L_{st} = 1))$

Table 1. Conditions for the decimal rounding modes implemented.

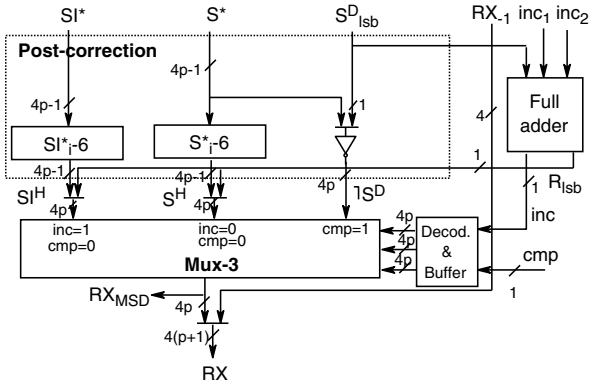

Figure 7. Post-correction and selection stages.

This is determined from S^*_{MSD} and SI^*_{MSD} (coded in BCD excess-6) as

$$MSD > 0 \Longleftrightarrow (S^*_{MSD} > 6) \vee (SI^*_{MSD} > 6)\ inc_1\ S_{lsb}$$

The comparisons > 6 are implemented examining if the msb or the lsb of S^*_{MSD} (resp. SI^*_{MSD}) are '1'. The critical path goes from SI^*_{MSD} to inc_2 (4 gate delays).

4.4. Post-correction and selection stages

The implementation of the post-correction stage (inside the dashed box) and the selection stage is shown in Fig. 7. The signal inc and R_{lsb} are computed in a full adder as the sum and carry bits of $inc_1 + inc_2 + S^D_{lsb}$. This full adder contributes with an XOR gate delay to the total delay of the critical path (inc_2 signal). Hence, the $4p-1$ most significant bits of RX are selected from the BCD sums S^H, $\neg S^H$ and SI^H using a level of 3:1 muxes as follows:

• If $cmp = eop\ \overline{C_{out}} = 1$ then $U - L < 0$ (no rounding required), and the magnitude result R is given by the 9's complement of $S^D = S^H + S^D_{lsb}$. Since the 9's complement of the BCD sum digits $S^H_i = S^*_i - 6$ is given by $\neg S^H_i = \overline{(S^H_i + 6)} = \overline{S^*_i - 6 + 6} = \overline{S^*_i}$, then $S^H = \overline{S^*}$.

• If $inc = 1$ and $cmp = 0$ then SI^H is obtained from the digits SI^*_i after a +6 digit subtraction, that is, $SI^H = \sum_{i=0}^{p-1}(SI^*_i - 6)\ 10^i$. The blocks labeled as '-6' in Fig. 7 implement this digit subtraction in 2 level of logic (1 XOR level).

• If $inc = 0$ and $cmp = 0$, then S^H is obtained from the S^*_i digits as $S^H = \sum_{i=0}^{p-1}(S^*_i - 6)\ 10^i$

The selection signals for the 3:1 muxes are decoded as cmp, $inc\ \overline{cmp}$ and $\overline{inc \vee cmp}$, and must be buffered due to the high load. The lsb of RX is obtained as $\overline{S^D_{lsb}}\ cmp \vee RX_{lsb}\ \overline{cmp}$ in a 2:1 mux. Finally, the guard digit RX_{-1} is concatenated to the right of the lsb to form the rounded $p+1$-digit significand RX.

5. Estimations and comparison

We use an evaluation model for static CMOS technology to estimate the area and delay of our architecture for the IEEE 754-2008 Decimal64 and Decimal128 formats and to compare with the corresponding architecture proposed in [12]. The delay model is based on a simplification of the logical effort method [8] that allows for faster hand calculations. In particular, it does not consider gate sizing optimizations. Instead, we assume gates with the drive strength of the 1x inverter (minimum sized gates), using buffers or cloning (gate replication) to drive high loads. Delay values are given in FO4 units (delay of an minimum sized (1x) inverter with a fanout of four 1x in-

Stage	IEEE Decimal64		IEEE Decimal128	
	Delay #FO4	Area Nand2	Delay #FO4	Area Nand2
Pre-correction	6.5	1010	6.5	2140
Binary adder	10.5	1800	12.7	4300
Rounding	2.5	200	2.5	200
Selection	6.6	570	7.4	1160
Total	**26.1**	**3580**	**29.1**	**7800**

Table 2. Area and delay estimations.

Adder	Delay # FO4	Ratio	Area Nand2	Ratio
IEEE 754-2008 Decimal64				
Proposed	26.1	1.00	3580	1.00
Wang& Schulte [12]	30.3	1.16	4490	1.25
IEEE 754-2008 Decimal128				
Proposed	29.1	1.00	7800	1.00
Wang& Schulte [12]	32.2	1.11	9950	1.28

Table 3. Comparison figures.

verters). To measure the total hardware cost in terms of number of gates, the area of the different gates is normalized using a reference gate, in this case a 1x two-input NAND gate (NAND2). Thus, the area of a design is estimated as the number of equivalent NAND2 gates.

5.1. Delay analysis and area estimation

The general block diagram of our architecture is depicted in Fig. 8. The critical path is indicated as a discontinuous thick line. In terms of logic levels (XOR gate=2 logic levels) the critical path goes through 6 levels of the pre-correction stage, $4+\log_2(4p)$ levels of the binary compound adder, 4 levels of the rounding logic, 1 level in the increment stage, and though a chain of buffers (with a load of $4p$ muxes) plus 2 levels of the selection stage. The total number of levels (excluding the buffering) is 23 for Decimal64 ($p = 16$) and 25 for ($p = 34$). In Table 2 we present the delay (in # FO4) and area (in # NAND2 gates) estimated using the area and delay evaluation model for the IEEE 754-2008 Decimal64 and Decimal128 implementations. The delay figures correspond with the critical path delay of each stage. For the binary compound adder, the figures correspond with a 63-bit Kogge-Stone prefix tree adder. The contribution of the post-correction logic to the area is included in the selection stage. Note that an important contribution to the delay of the selection stage is due to the buffering chain. In addition, most of the hardware cost comes from the pre-correction stage. For instance, the implementation of G_i costs about 33 NAND2 gates per digit, while the cost of the block $+6$ G_i is about 18 NAND2 gates per digit. However, this simplifies significatively the logic required to merge rounding with significand addition and the decimal post-correction of the binary sum.

5.2. Comparison

First, we examine the implementation proposed in [12] for merging significand BCD addition with decimal rounding. Although the implementation was originally proposed for the Decimal64 format (16 digits), we also consider an implementation for Decimal128 (34 digits). It uses a sign-magnitude BCD adder and a decimal variation of the rounding by injection algorithm [7]. The sign-magnitude BCD adder ($SR = |U + (-1)^{eop}L + inj|$) is implemented using a $4(p+3)$-bit binary flagged prefix adder [1] with decimal pre and post corrections. The BCD magnitude SR is obtained in a complex post-correction unit selecting the $p + 3$-digit BCD sums S, $\neg S$ and $SI = S + 1$ obtained from the binary sum. This selection depends on C_{out} and eop. We estimate a cost of 85 NAND2 gates per digit and 8 levels of logic for this unit. When the MSD of SR is not zero, an injection correction value inj_{cor} is inserted into a 2-digit BCD adder. It produces a carry-out inc_2 to the LSD of SR and this may generate a decimal carry propagation from the LSD of SR. The length of this propagation is determined by the trailing chain of 9's of SR. To reduce the critical path delay of the unit, the trailing 9's detection is overlapped with the BCD post-correction: a pair of trees of AND gates (one for addition and one for subtraction) compute a set of flags $f2_i^+$ (for $eop = 0$) and $f2_i^-$ (for $eop = 1$) from the binary sum digits S_i^* and the decimal carries C_i. The digits of RX are selected from $SR_i + 1$ (digitwise increment) or SR_i, depending on these flags and the signal inc_2.

Summarizing, the critical path goes through 2 logic levels for the pre-correction, $3 + \log_2(4p + 12)$ levels of the binary compound adder, 8 logic levels of the post-correction unit, 3 levels of the rounding logic, a chain of buffers loading $4p$ muxes and 2 levels for the final selection. This gives a total delay of 28 logic levels for Decimal64 and 29 logic levels for Decimal128. Using our model, we have evaluated the area and delay of this architecture for both Decimal64 and Decimal128 formats. Table 3 shows these results and presents the corresponding ratios with respect to our proposals. We estimate improvements between 11% and 16% in performance and area reductions of 25% and 28% for the architecture proposed in this paper.

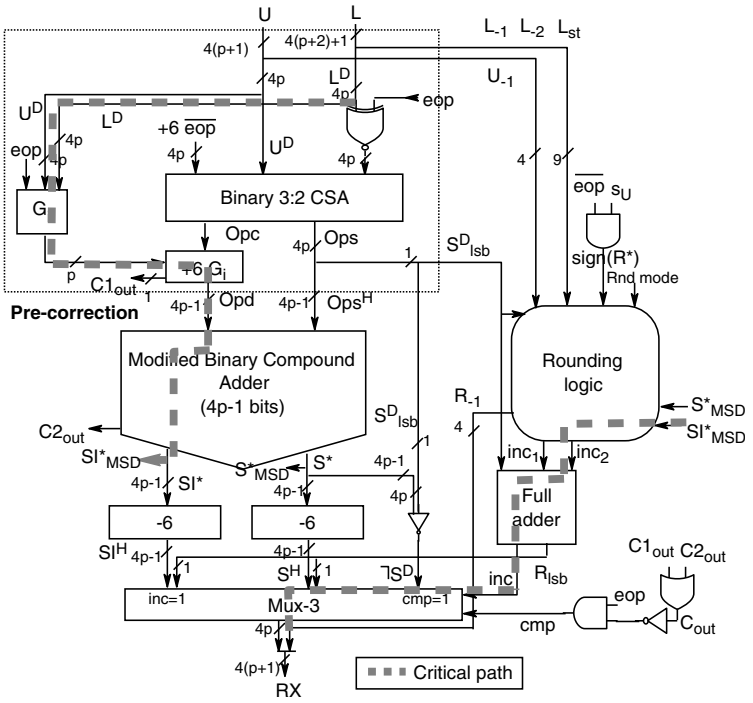

Figure 8. Proposed sign-magnitude BCD adder with rounding.

6. Conclusions

A new method and the architecture for merging significand BCD addition and decimal rounding were presented. This is of interest to improve the efficiency of high-performance IEEE 754-2008 DFP units. Furthermore, the IEEE 754-2008 Decimal64 (16 precision digits) and Decimal128 (34 precision digits) implementations presents speedups of 16% and 11% in performance while reduce the area of significand computation and rounding more than 25% with respect to a previous representative high-performance DFP adder [12]. This significant reduction in area lie in a simplification of the logic for decimal post-correction and rounding.

References

[1] N. Burgess. Prenormalization Rounding in IEEE Floating-Point Operations Using a Flagged Prefix Adder. *IEEE Trans. on VLSI Systems*, 13(2):266–277, Feb. 2005.

[2] M. F. Cowlishaw. Decimal Floating-Point: Algorism for Computers. In 16^{th} *IEEE Symposium on Computer Arithmetic*, pages 104–111, Jul. 2003.

[3] L. Eisen et al. IBM POWER6 accelerators: VMX and DFU. *IBM Journal of Research and Development*, 51(6):663–684, Nov. 2007.

[4] IEEE Std 754(TM)-2008. *IEEE Standard for Floating-Point Arithmetic*. IEEE Computer Society, Aug. 2008.

[5] R. K. Richards. *Arithmetic Operations in Digital Computers*. D. Van Nostrand Company, New Jersey, 1955.

[6] E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw. Decimal Floating-Point Support on the IBM System z10 processor. *IBM Journal of Research and Development*, 51(1):97–113, Jan/Feb. 2009.

[7] P. M. Seidel and G. Even. Delay-Optimized Implementation of IEEE Floating-Point Addition. *IEEE Trans. on Computers*, 53(2):97–113, Feb. 2004.

[8] I. Sutherland, R. Sproull, and D. Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann, 1999.

[9] J. Thompson, N. Karra, and M. J. Schulte. A 64-bit Decimal Floating-Point Adder. In *IEEE Computer Society Annual Symposium on VLSI*, pages 297–298, Feb. 2004.

[10] A. Vazquez. *High-Performance Decimal Floating Point Units*. PhD thesis, Univ. of Santiago de Compostela, Mar. 2009. http://www.ac.usc.es/biblio/keyword/PhDThesis.

[11] A. Vazquez and E. Antelo. Conditional Speculative Decimal Addition. In 7^{th} *Conference on Real Numbers and Computers (RNC 7)*, pages 47–57, Jul. 2006.

[12] L.-K. Wang and M. J. Schulte. Decimal Floating-Point Adder and Multifunction Unit with Injection-Based Rounding. In 18^{th} *IEEE Symposium on Computer Arithmetic*, pages 56–68, Jun. 2007.