Efficient Decimal Leading Zero Anticipator Designs

Mohamed H. Amin, Ahmed M. Eltantawy, Alhassan F. Khedr, Hossam A. H. Fahmy, Ahmed A. Naguib

***Abstract*— The leading zero anticipator (LZA) is a vital block in fast floating point addition and fused multiply-add (FMA) operations. So far, there is only one decimal LZA proposed in research literature. This paper introduces two decimal LZA designs, then a comparison between the three designs, the two proposed here and the previous proposed one, is performed.**

Index Terms--- LZA, LZD, floating point, Decimal, FMA

I. INTRODUCTION

Although the binary system is widely used in computer arithmetic designs, the need for the decimal computing arises in many applications such as financial and scientific ones. Some applications use the decimal processing in 50% to 90% of their work [1]. Executing these applications in software adds more delay as software libraries are much slower than hardware designs (by a factor of 100x to 1000x performance degradation) [1] [2]. This explains the need of the decimal floating point (DFP) hardware units.

Since the inclusion of the DFP specifications in IEEE 754-2008 standard [3], many architectures have been introduced for the DFP addition [4] [5], multiplication [6] [7] [8], and division [9] [10]. The only implemented decimal fused multiply-add (DFMA) is introduced in [11] and a proposed general design without implementation is introduced in [12].

One of the basic units in FP addition and FMA operations is the leading zero detector (LZD). It waits for the result of the adder to count the number of its leading zeros. This count is then used to left shift the result to save precision and meet standard specifications. The LZD increases the critical path delay as shown in Figure 1(a). Replacing the sum LZD by a leading zero anticipator (LZA) is vital to increase the performance of FP processors [13]. The LZA anticipates the leading zeros count of the result directly from the input operands (with possible error of one bit in binary or one digit in decimal). It works in parallel with the adder, so it eliminates the leading zero detection from the critical path, Figure 1(b).

Although many binary LZAs were introduced [14]-[19], the only decimal LZA is proposed by [4].

This paper is organized as follows. Section II surveys the previous work on LZA in both binary and decimal. Section III introduces two proposed decimal LZA designs. A comparison between the three decimal LZA designs, the two proposed here and that in [4], is performed in section IV. Finally, the results of the work are concluded in section V.

II. PREVIOUS WORK

The LZA anticipates the leading zero count of the sum from the input operands. This anticipation may have an error of one bit in binary system or one digit in decimal system. In this section we try to survey the attempts to correct this error both in binary and in decimal.

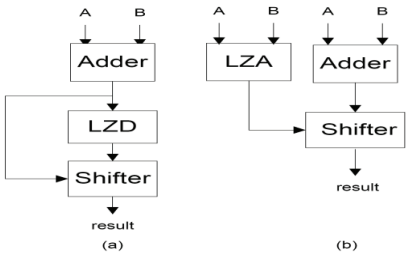

Figure 1: (a) LZD (b) LZA

A. Binary LZA Correction Designs

Five main architectures are proposed in binary to correct this error without adding a large delay to the critical path.

The first one [14] is shown in Figure 2(a). It anticipates the number of leading zeros by the LZA and decodes it to feed the coarse shifter. The normalization operation is composed of two shifters, the coarse shifter and the fine shifter. The one-bit error is corrected by adding a compensation shifter after the normalization stage. This compensation shifter is in the critical path and has an estimated delay of 0.8 ns (on 0.5µm CMOS technology) which is 10% of the total LZA delay [14].

As the possible error in one bit depends on the carry to this bit, the second architecture proposed in [15], [16] waits for the carries from the adder to check the predicted LZC by a carry select circuit to feed the shift correction circuit. This corrected shift affects only the least significant bits in the normalization stage and hence detection and correction are in parallel with the normalization coarse shifter, but this correction circuit is in the critical path as shown in Figure 2(b).

The third one is introduced in [19] and shown in Figure 2(c). It uses a LZA that generates a one-hot vector has the same number of bits as the sum and has a '1' at the bit directly adjacent to the leading zero bits (the leading one bit). It corrects the error by a correction circuit which compares the one-hot vector with the sum in parallel with the coarse shifter. This comparison is simply a stage of AND gates followed by an ORing stage. The comparison result which indicates whether there is an error or not feeds the fine shifter to produce the final normalized output.

The fourth architecture shown in Figure 2(d) [18] depends on comparing the position of the leading one of the predicted LZC with that of the sum. This leading one may be either in an odd or even bit position. If this oddness is the same in both the sum and the predicted LZC, the predicted LZC is correct; otherwise, correction is needed. Although this design has the same general block diagram of [19], the correction circuit is out of the critical path. This is because comparing the oddness feature of the leading one is simpler and faster than comparing the whole predicted vector and the sum, and it is sufficient as the prediction error lies only in one bit.

The fifth architecture shown in Figure 2(e) was introduced by Bruguera and Lang [17]. It uses a detection tree in parallel

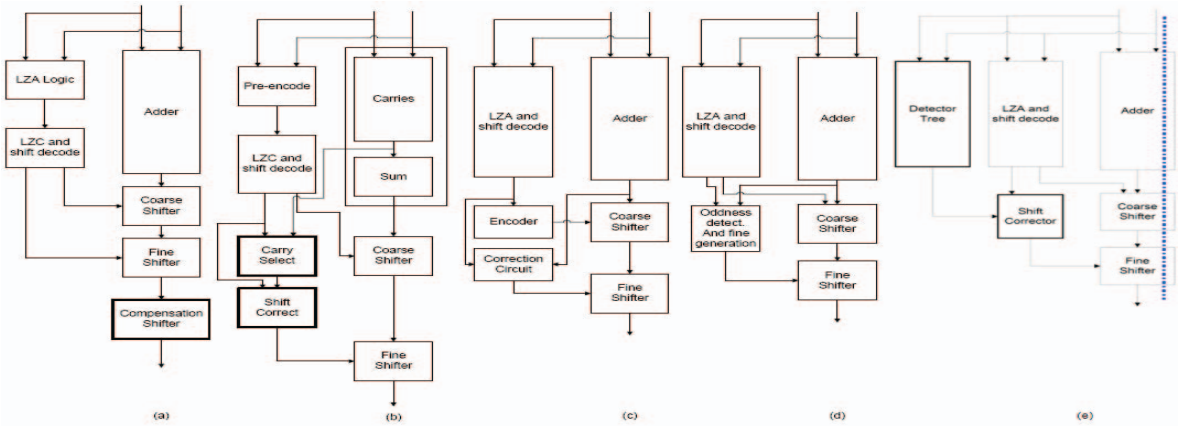

Figure 1 LZA based on (a) compensation shifter correction. (b) carry select correction (c) one hot vector (d) oddness detection (e) parallel tree

with the prediction circuit. This design has a smaller delay as the detection tree is out of the critical path, but it has the disadvantage of the large consumed area. It has 80% larger area than the LZA without parallel detection tree (first and second architectures) [17].

B. Decimal LZA based on Correction Tree

Wang and Schulte are the only authors to our knowledge who introduced a decimal LZA [4]. It is shown in Figure 3, and based on the correction tree idea of [17].

Unlike binary, in decimal we have to determine the LZC of the result in both effective addition and effective subtraction. This is because the operands in decimal are not normalized.

They divided the LZA into two completely separate parts, one for effective addition and the other for effective subtraction. Finally, the effective operation signal *eop* chooses the correct one. These two LZAs are preceded by a parallel array of 16 BCD Adders.

In effective addition LZA the LZC is calculated as:

$$LZC_{Add} = minimum\ (LZC_A, LZC_B) - Y_{add} \qquad (1)$$

LZC_A, LZC_B are the leading zero counts for the two inputs, and Y_{add} is the correction signal.

The correction signal $Y_{add} = 1$ if the pattern $zm^x\, pm^y\, gm^z$ is detected. gm indicates a carry generate digit, pm indicates a carry propagate digit, and zm indicates a zero digit. The three signals are calculated for each sum digit. A 4-level tree (for decimal64) is used to calculate Y_{add} in parallel with the prediction.

In effective subtraction there are two cases ($sum > 0, sum < 0$). Wang and Schulte followed [17] in using only one anticipation unit and two separate correction trees. The anticipation unit generates a binary string P with the same LZC as the sum (with possibly an error of one digit to the left or the right). The two correction trees generate two correction signals, one for each case ($Y_{sub,pos}$, $Y_{sub,neg}$). Finally, they use the $sign$ signal to choose one of the two correction signals to get the final effective subtraction correction signal Y_{sub}.

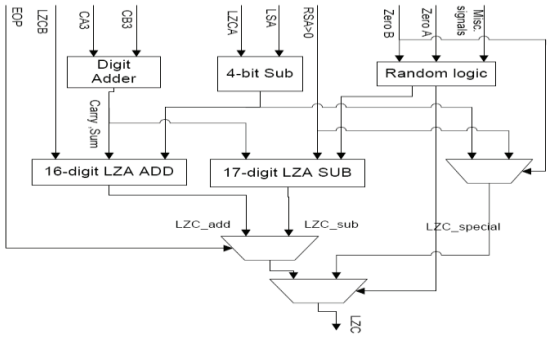

Figure 3: Decimal64 LZA in [8]

Table 1: Signals in effective subtraction LZA

Signal	A
$g9$	$B+9$
$g2$	$[B+2, B+9]$
$g1$	$B+1$
$zero$	B
$s1$	$B-1$
$s2$	$[B-9, B-2]$
$s9$	$B-9$
$g9$	$g9$

Table 3: CLA generated signals

Case	P_i	G_i	Z_i
Effective sub.	$Zero_i$	Gr_i	—
Effective Add.	pm_i	gm_i	zm_i

Table 2: Signals in effective addition LZA

Signal	Condition		
	Comparator	CLA	
p_m	$A+B=9$	$A+B=15$	Carry propagate
g_m	$A+B \geq 10$	$A+B \geq 15$	Carry generate
z_m	$A+B=0$	$A+B=6$	zero

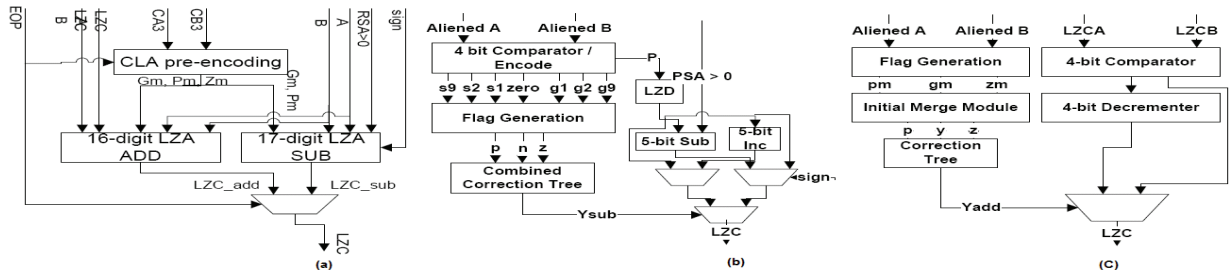

Figure 4: (a) LZA using the CLA pre-encoding (b) Effective Subtraction LZA (c) Effective Addition LZA

III. Proposed Designs

This paper introduces two decimal LZA designs. One is based on the parallel detection tree method and the other is based on the oddness detection method. Both methods were described in section II.

A. Decimal LZA based on Parallel Detection Tree

The first Design is adapted from [4]. It optimizes the area and the speed as will be shown in the comparison section. Unlike [4] which is designed for a special adder design and assumes decoded operands, this is a general LZA that can be used for FP addition and FMA designs which follow the IEEE floating point standard [3]. It consists of three main blocks, the pre-encoding unit, the effective subtraction LZA, and the effective addition LZA.

1. The Pre-encoding Unit

The goal of the pre-encoding is to generate the signals needed for the subtraction and addition trees as shown in Table 1 and Table 2. We implemented this unit using two mechanisms: CLA concept and direct comparisons

CLA Pre-encoding

It uses the same concept of the Carry Look- Ahead (CLA) of generating the carry in a faster way (estimate of 3 gate delays for one digit).

It generates the signals propagate g_j and generate p_j for each bit $j \in [0,3]$ in the digit i where:

$$g_j = A_j . B_j$$
$$p_j = A_j \oplus B_j$$

We assume here that the inputs A and B are pre-corrected inputs, i.e. in the subtraction case A_i is not changed and $B_i = 15 - CB_i$, while in the addition case B_i is not changed and $A_i = CA_i + 6$, CA_i and CB_i are the significants of the two inputs after the alignment. For each digit i we have:

$$P_i = p_0 p_1 p_2 p_3$$

$$G_i = g_3 + g_2 . p_3 + g_1 . p_2 . p_3 + g_0 . p_1 . p_2 . p_3 + P_i . eop$$

$$Z_i = g_0' p_0' p_1 p_2 p_3' g_3' + g_0 g_1 p_2' g_2' p_3' g_3' + g_0 g_1' p_1' p_2 p_3' g_3$$

These three signals have different meanings in effective addition and effective subtraction cases. In effective subtraction: $P_i = 1$ means that both digits are equal, and $G_i = 1$ means that there is a carry from the subtraction which indicates that $A_i > B_i$. In effective addition, they are used by their direct as shown in Table 3. The LZA with the CLA pre-encoding is shown in Figure 4 (a).

Comparator Pre-encoding

In this mechanism the pre-encoding stage will have two units, one for effective subtraction and the other for effective addition (as shown in Figures 4(b) and 4(c)). ***In effective subtraction***, an array of 16 four-bit comparators is used to compare each digit of the two aligned inputs and hence generate the signals indicating in which category is the input A with respect to the second input B. Table 1 shows the condition on A to set each signal.

In effective addition, we also use the two operands directly to generate the signals (pm, gm, zm) shown in Table 3.

Sign Detection

In [4], Wang and Schulte assumed that the sign needed in correction (as illustrated in Section II (B)) is detected after the addition. However, it is not applicable in some architectures, for example to use a combined add/round block the shift must be performed prior to addition. So, the LZA cannot wait for the addition result. Hence, we propose a simple sign detection tree that operates in parallel with the correction tree and the anticipation. We use the comparator to generate the vector Gr where $Gr_i = 1$ indicates that digit A_i is greater than digit B_i. This vector and the zero vector are used as inputs to the sign detection tree shown in Figure 5.

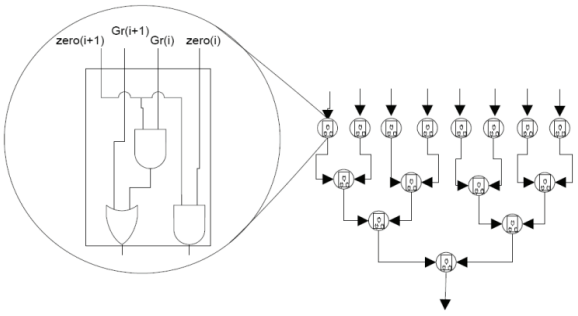

Figure 5: Sign Detection Tree

Effective Addition LZA

The effective addition LZA shown in Figure 4(c) depends on the comparator-based pre-encoding. Its architecture is very similar to [4] as it is straight forward, can be used generally, and finally it consumes a small area (less than 10% of the total LZA area).

The anticipation unit uses equation (1) to get the predicated effective addition LZC LZC_{Add}.

In parallel with this anticipation, the initial merging module uses the signals generated by the pre-encoding unit (gm, pm, zm) to generate the arrays' initial values $(z_{0,i}, p_{0,i}, y_{0,i})$ as illustrated in equations (2). These arrays enter the tree logic of equations (3) to finally get the correction signal Y_{add} [4].

$$\begin{aligned} z_{0,i} &= zm_i . zm_{i-1} \\ p_{0,i} &= (zm_i + pm_i) . pm_{i-1} \\ y_{0,i} &= gm_i + (zm_i + pm_i) . gm_{i-1} \end{aligned} \quad (2)$$

$$\begin{aligned} z_{L,i} &= z_{L-1,2i+2^{L-1}} . z_{L-1,2i} \\ p_{L,i} &= \left(z_{L-1,2i+2^{L-1}} + p_{L-1,2i+2^{L-1}}\right) . p_{L-1,2i} \\ y_{L,i} &= y_{L-1,2i+2^{L-1}} + \left(z_{L-1,2i+2^{L-1}} + p_{L-1,2i+2^{L-1}}\right) . y_{L-1,2i} \end{aligned} \quad (3)$$

Finally,

$$Y_{add} = y_{4,0}$$

Effective Subtraction LZA

The effective subtraction LZA shown in Figure 4(b) consists of two parallel units: the anticipation unit and the detection unit. In effective subtraction we have two possibilities (A>B) and (B<A). The anticipation unit is combined for both cases as shown in equation (4). The detection unit consists of the Flag Generation module and the tree. The Flag Generation module uses the pre-encoded signals to find out the arrays needed by the tree for the positive sum (equations (5)) and the negative sum (equations (6)). We combine them in equations (7) using the sign signal produced from the sign detection module to have the arrays required for the tree (p, n, z) Where p indicates a leading one digit, z indicates a zero digit, n indicates a leading zeros terminate digit. These combined vectors enter only one tree (equations (8)) to get the effective subtraction correction signal Y_{sub}. The combined tree has the same area as only one correction tree in [4] which saves the area of a complete correction tree (the correction tree has an estimate of 150 gate count) without any additional cost on delay.

$$\begin{aligned} P_i &= zero_{i+1} . (g2_i + s2_i + g1_i . s9'_{i-1} + s1_i . g9'_{i-1}) \\ &+ zero'_{i+1} . (s9_i . s9'_{i-1} + g9_i . g9'_{i-1}) \end{aligned} \quad (4)$$

$$(sum > 0)$$

$$\begin{aligned} p_{po\,s_i} &= (g1_i + s9_i) . s9'_{i-1} \\ n_{po\,s_i} &= zero_{i+1} . s9_i + s1_i + s2_i . s9'_i \\ z_{po\,s_i} &= zero_i + (g1_i + s9_i) . s9_{i-1} \end{aligned} \quad (5)$$

$$(sum < 0)$$

$$\begin{aligned} p_{ne\,g_i} &= (s1_i + g9_i) . g9'_{i-1} \\ n_{po\,s_i} &= zero_{i+1} . g9_i + g1_i + g2_i . g9'_i \\ z_{po\,s_i} &= zero_i + (s1_i + g9_i) . g9_{i-1} \end{aligned} \quad (6)$$

$$\begin{aligned} p_i &= p_{ne\,g_i} . sign + p_{po\,s_i} . sign' \\ n_i &= n_{ne\,g_i} . sign + n_{po\,s_i} . sign' \\ z_i &= z_{ne\,g_i} . sign + z_{po\,s_i} . sign' \end{aligned} \quad (7)$$

$$\begin{aligned} Tz_{L,i} &= Tz_{L-1,2i+2^{L-1}} . Tz_{L-1,2i} \\ Tp_{L,i} &= Tz_{L-1,2i+2^{L-1}} . Tp_{L-1,2i} + Tp_{L-1,2i+2^{L-1}} . Tz_{L-1,2i} \\ Tn_{L,i} &= Tz_{L-1,2i+2^{L-1}} . Tn_{L-1,2i} + Tn_{L-1,2i+2^{L-1}} \\ Ty_{L,i} &= Ty_{L-1,2i+2^{L-1}} + Tz_{L-1,2i+2^{L-1}} . Ty_{L-1,2i} \\ &\quad + Tp_{L-1,2i+2^{L-1}} . Tn_{L-1,2i} \end{aligned} \quad (8)$$

Where in our combined tree:

$$Tp_{0,i} = p_i, Tn_{0,i} = n_i, Tz_{0,i} = z_i, Ty_{0,i} = 0$$

and finally,

$$Y_{sub} = y_{4,0}$$

B. Decimal LZA based on Oddness Detection

The binary oddness correction technique is modified to work on decimal. In decimal we need only to indicate whether each digit is zero or not, so each 4-bit digit can be represented by only one bit, this is done by a parallel OR gates before the oddness detection then perform the oddness detection by the same circuit used in [18]. Figure 6 shows this detection circuit for decimal64.

We can see that for Decimal64 we need only three levels of the A and B blocks to generate the correction signal, while in Binary64 we need five levels to generate it which means that the decimal oddness circuit will be faster. Also, it will consume a less area which makes this technique very suitable to decimal. This correction can be performed in parallel with coarse shifters which operate on the preliminary value of LZC, thus it adds no delay to the critical path of the adder. However, the dependency of the correction on the sum limits the flexibility of the architecture.

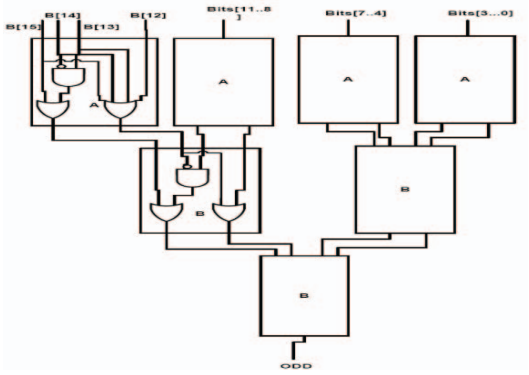

Figure 6: Decimal64 Oddness Detection Circuit

IV. COMPARISONS

Three LZA designs were implemented in VHDL (the previous proposed one and the two proposed here). Then, they are synthesized by the Cadence RTL Compiler using TSMC65LP Technology. Table 4 compares the one in 3.1, with its two pre-encoding implementations, with that of [4] as they depend on the same idea. The results show that at the same area($5189\mu m^2$) , the proposed design has 17.7% delay

improvement in CLA pre-encoding based design, and 15.3% delay improvement in Comparator pre-encoding based design. At the same delay ($0.85nm$), the proposed LZA has 22% less area in CLA pre-encoding based design, and 24.2% less area in Comparator pre-encoding based. Also the critical path delays on best performance case shows that the LZA in [4] has 24 FO4 inverter delays, while the proposed one has 19 FO4.

The results show also that the CLA pre-correction LZA has less delay than the Comparator pre-correction LZA, while the second has less area. This is logical as the CLA unit generates the carry in a faster way which improves the delay. However, it consumes large area. So, if the LZA will be in the critical path of the total architecture (FMA or adder), it is recommended to use the CLA pre-encoding based design; otherwise use the Comparator pre-encoding based design to save the area.

Table 4: Comparison between different decimal LZA designs

	LZA in [4]	Proposed LZA	
		CLA pre-enc.	Comparator pre-enc
Delay	0.85nm	0.7nm	0.72nm
Area	$5189\mu m^2$	$4050\mu m^2$	$3932\mu m^2$

Regarding the proposed LZA based on the oddness detection, we synthesized the preliminary LZA circuit and the oddness detection circuit separately. This is because the nature of the design discussed in section III (B). Table 5 shows the synthesis results. These results show that this design has the advantage of very low area relative to the Parallel-Correction based ones.

Table 4: Optimized results for Oddness-Detection based LZA

	preliminary LZA	Oddness Detect. and Corr.
Area opt.	$1108.8\mu m^2$	$58.7\mu m^2$
Delay opt.	0.28nm	0.1nm

V. CONCLUSION

Two new decimal LZA were proposed in this paper. One of them is based on the oddness-detection idea, while the other is based on the correction tree idea. The oddness based one is very efficient in area and is not in the critical path, but it has a limitation on design flexibility as the correction must wait the addition result. The correction- tree based LZA proposed here is implemented with two different pre-encoding mechanisms. It decreases the area by more than 20% at the same delay of the only previous proposed one. It also decreases the delay by more than 15% at the same area.

REFERENCES

[1] M. F. Cowlishaw. Decimal floating-point: Algorism for computers. In Proc. IEEE 16th Symposium on Computer Arithmetic, pages 104–111, July 2003.

[2] L.-K. Wang, C. Tsen, M. J. Schulte, D. Jhalani, "Benchmarks and Performance Analysis for Decimal Floating-Point Applications," IEEE International Conference on Computer Design, October 2007.

[3] IEEE Standard for Floating-Point Arithmetic, IEEE Std. 754-2008, Aug. 2008. 1–58.

[4] L.-K. Wang, M. J. Schulte, A Decimal Floating-Point Adder with Decoded Operands and a Decimal Leading-Zero Anticipator, 19th IEEE Symposium on Computer Arithmetic, pages 125 - 134 , June 2009.

[5] L.-K. Wang, M. J. Schulte, J. D. Thompson, and N. Jairam, "Hardware designs for decimal floating-point addition and related operations," IEEE Transactions on Computers, vol. 58, no. 3, pp. 322–335, Mar. 2009.

[6] M. A. Erle, M. J. Schulte, and B. J. Hickmann, "Decimal floating-point multiplication via carry-save addition," in Proceedings of the IEEE International Symposium on Computer Arithmetic, 25–27 June, 2007, Montpellier, France.

[7] R. Raafat, A. M. Abdel-Majeed, R. Samy, T. ElDeeb, Y. Farouk, M. Elkhouly, and H. A. H. Fahmy, "A decimal fully parallel and pipelined floating point multiplier," in Forty-Second Asilomar Conference on Signals, Systems, and Computers, Asilomar, California, USA, Oct. 2008.

[8] A. Vazquez, E. Antelo and P. Montuschi, "Improved Design of High-Performance Parallel Decimal Multipliers," IEEE Transactions on Computers, Vol. 59, No. 5, pp. 679-693, May 2010.

[9] L.-K. Wang and M. J. Schulte, "A decimal floating-point divider using Newton–Raphson iteration," Journal of VLSI Signal Processing, vol. 49, no. 1, pp. 3–18, Oct. 2007.

[10] H. Nikmehr, B. Phillips, and C.-C. Lim, "Fast decimal floating-point division," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 14, no. 9, pp. 951– 961, Sep. 2006.

[11] R. Samy, H. A. H. Fahmy, R. Raafat, A. Mohamed, T. ElDeeb, Y. Farouk, A Decimal Floating-Point Fused-Multiply-Add Unit, 53rd IEEE International Midwest Symposium on Circuits and System, Pages 529 - 532, Aug. 2010.

[12] A. Vázquez. High-Performance Decimal Floating-Point Units. Ph.D. thesis, Universidade de Santiago de Compostela, 2009.

[13] M. S. Schmookler and K. J. Nowka. Leading Zero Anticipation and Detection – A Comparison of Methods. In Proceedings of the 15th IEEE Symposium on Computer Arithmetic, pages 7–16, June 2001.

[14] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko, T.Sumi, "Leading-zero Anticipatory Logic for High-speed Floating Point Addition", IEEE Journal of Solid State Circuits, August , pp.1157-1164, 1996.

[15] E. Hokenek and R. Montoye, "Leading-Zero Anticipator (LZA) in the IBM RISC System/6000 Floating Point Execution Unit", IBM Journal of Research and Development, pp. 71-77, 1990.

[16] N. Quach and M. Flynn, "Leading One Prediction – Implementation, Generalization, and Application", Technical Report CSL-TR-91-463, Stanford University, March, 1991.

[17]J. Bruguera, E .Lang "Leading-One Prediction with Concurrent Position Correction" IEEE Transactions on Computers, v. 48, No. 10, October 1999, pp. 298-305.

[18] C.N. Hinds and D.R. Lutz. A Small and Fast Leading One Predictor Corrector Circuit. In Proc. 39 Asilomar Conference on Signals, Systems and Computers., pages 1181–1185. IEEE, 2005.

[19] R Rogenmoser and L. O'Donnell, "Method and apparatus to correct leading one prediction," US Patent application 2002-0165887, Nov 2002.