Multi-operand Decimal Addition by Efficient Reuse of a Binary Carry-save Adder Tree

Álvaro Vázquez, and Elisardo Antelo

Multi-Operand Decimal Addition by Efficient Reuse of a Binary Carry-Save Adder Tree

Álvaro Vázquez
INRIA - Laboratoire LIP, CNRS-ENSL-INRIA-UCBL
Ecole Normale Supérieure de Lyon
46 Allée d'Italie
69364 Lyon Cedex 07, France
Email: Alvaro.Vazquez-Alvarez@inrialpes.fr

Elisardo Antelo
Department of Electronic and Computer Engineering
University of Santiago de Compostela
Edificio Monte da Condesa, Campus Sur
15782 Santiago de Compostela, Spain
Email: elisardo.antelo@usc.es

***Abstract*—We present a novel method for hardware design of combined binary/decimal multi-operand adders. More specifically, we apply this method to architectures based on binary CSA (carry-save adder) trees, which are of interest for VLSI implementation of high performance multipliers and other low latency arithmetic units. A remarkable feature of the proposed method is that it allows the reuse of any binary CSA for computing the sum of BCD operands. Decimal corrections are performed in parallel, separately from the computation of the binary sum, such that the layout of the binary carry-save adder does not require any further rearrangement. As a result, the latency of the binary operation is unaffected by the incorporation of hardware support for decimal, while the latency for the decimal mode is close to the latency figures of dedicated decimal multi-operand adders. We show that our combined architecture is competitive in terms of area and delay with respect to other representative proposals, and that it has a more regular layout when implemented in a submicron VLSI technology.**

I. Introduction

Decimal floating-point units are currently being integrated in high-end processors to accelerate financial, commercial and user-oriented server applications [5]. However, a widespread use of decimal hardware for general purpose computation is not expected in a near future since they present significant area and latency overheads with respect to binary units. An alternative could be to replace the binary units of current microprocessors by equivalent combined binary/decimal units, which should have similar latencies for binary operations but at a moderate hardware cost overhead.

Fast multi-operand addition is of key importance to implement low latency multipliers and other basic fixed and floating-point arithmetic units. For example, to provide support for IEEE 754-2008 Decimal64 multiplications [3], parallel implementations of BCD multipliers must be able to reduce up to 17 or 32 decimal operands [8]. A different approach to implement IEEE 754-2008 decimal multiplication is to use binary for significand multiplication and additional hardware for decimal rounding and normalization [6]. Decimal multi-operand addition is more complex to implement than binary due to issues related to the inefficiency of decimal representations in boolean circuits. Several methods have been proposed to improve the efficiency of a straightforward implementation with BCD operands. An interesting proposal is the use of a different set of decimal codings than BCD (namely, 4221 and 5211) to represent decimal operands, allowing a decimal three-operand addition to be performed as a binary 3:2 carry-save addition with a small correction [7], [8].

In this paper we describe a method to implement efficient combined binary/decimal multi-operand adders based on these codes. Previous related proposals implement combined multi-operand addition either by introducing additional logic in the critical path of a binary carry-save adder tree [7], [2], [1], or by splitting the carry-save adder tree into shared, binary and decimal portions [2], [1]. Both solutions present area savings with respect to separate decimal and binary adder trees, but have a significant impact on the latency of the binary operation or lead to an irregular adder tree organization.

In our case, the whole binary carry-save adder tree is shared for both binary and decimal operations, which translates into additional area and power consumption savings and a more regular VLSI layout. Besides, the latency of the binary operation is unaffected by the incorporation of hardware support for decimal. This additional hardware computes a decimal correction amount which is added to the binary sum to produce the correct decimal result. Since the most part of this computation is overlapped with the binary carry-save addition, the maximum overhead delay of decimal multi-operand addition is bounded approximately by 10 XOR gate delays. Unlike the split tree presented in [2], [1], in our design decimal correction is completely separated from the binary carry-save adder, so that decimal hardware can be easily turned off to reduce power consumption in binary operation mode. Furthermore, it has a very regular and simple structure, which facilitates the integration of the proposed method into a CAD tool for automatic synthesis.

The structure of the paper is as follows. Section II gives an overview of related work. We provide a comprehensive description of the method for combined binary/decimal multi-operand addition in Section III. Section IV shows the proposed architecture. Section V presents some area and delay estimates and a comparison with other representative proposals. Conclusion and future work are presented in Section VI. An extended report can be found in [9].

Table I: Decimal codings

X_i	$X_i(BCD)$	$X_i(4221)$	$X_i(5211)$
0	0000	0000	0000
1	0001	0001	0001 0010
2	0010	0100 0010	0100 0011
3	0011	0101 0011	0101 0110
4	0100	0110 1000	0111
5	0101	0111 1001	1000
6	0110	1010 1100	1010 1001
7	0111	1011 1101	1011 1100
8	1000	1110	1110 1101
9	1001	1111	1111

II. Related work

An early binary/BCD multi-operand adder tree was proposed by Kenney and Schulte [4]. Input binary or BCD operands are directly passed to a binary carry-save adder tree to produce a two-word binary sum. Since only 10 out of the 16 possible combinations of 4 bits are legal BCD values, a correction is required to obtain the BCD sum from the binary sum. Besides, a +6 must added to a digit position each time a decimal carry-out is produced at that position. Both issues complicate the correction logic and have a significant impact on the hardware cost and the latency of the unit for decimal operation mode.

The decimal correction logic is simplified if the intermediate decimal operands are not represented in BCD, but in the 4221 or 5211 decimal codes shown in Table I, as proposed by Vazquez et al. [7], [8].

These decimal codes represent a digit X_i of a decimal integer operand $X = \sum_{i=0}^{p-1} X_i \ 10^i$ as a positive weighted 4-bit vector, that is, $X_i = \sum_{i=0}^{3} x_{i,j} \ r_j$, where $X_i \subset [0, 9]$ is the i^{th} decimal digit, $x_{i,j}$ is the j^{th} bit of the i^{th} digit and $r_j \geq 1$ is the weight of the j^{th} bit (BCD is obtained with $r_j = 2^j$). Codes 4221 and 5211 are of special interest for fast decimal carry-save addition: since all the sixteen 4-bit vectors represent a valid decimal digit ($X_i \in [0, 9]$), any boolean function operating at each weight bit produce the correct 4-bit vector representation of the result. Code 4221 is preferred than 5211 since it allows a faster conversion from BCD and to BCD.

Therefore, a decimal three-operand carry-save addition $X[1] + X[2] + X[3] = S + C \times 2$ can be implemented using a binary 3:2 carry-save adder followed by a $\times 2$ multiplication on the carry operand C [7], [8]. This multiplication $2C = C \times 2$ is implemented as a single level of decimal digit recoders, which map a digit C_i from $C_i(4221)$ to $C_i(5211)$ as shown in Table I, followed by a 1-bit left shift of the output [1]. Note that there are several ways to implement the logic of the digit recoders due to the redundancy of the 4221 and 5211 codes. The block diagram of a 1-digit (4-bit) decimal 3:2 carry-save adder is detailed in Fig. 1. The blocks labeled as **FA**, **3:2**, and $\times 2$ are respectivelly full adders with a fast input (of 1 XOR delay, indicated with an **F**), 4-bit binary 3:2 carry save adders,

[1]This 1-bit left shifting transforms the weight bits 5211 into weight bits 10 422, such that the digits of $2C$ are represented in code 4221

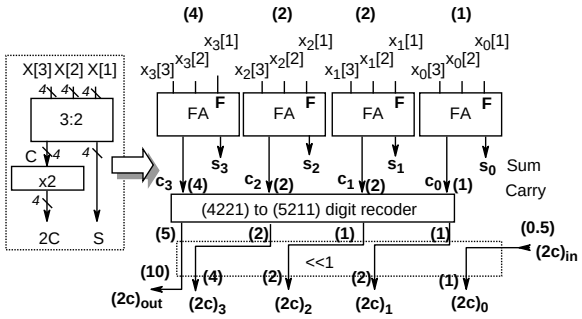

Figure 1: Decimal 3:2 carry-save adder (1 digit)

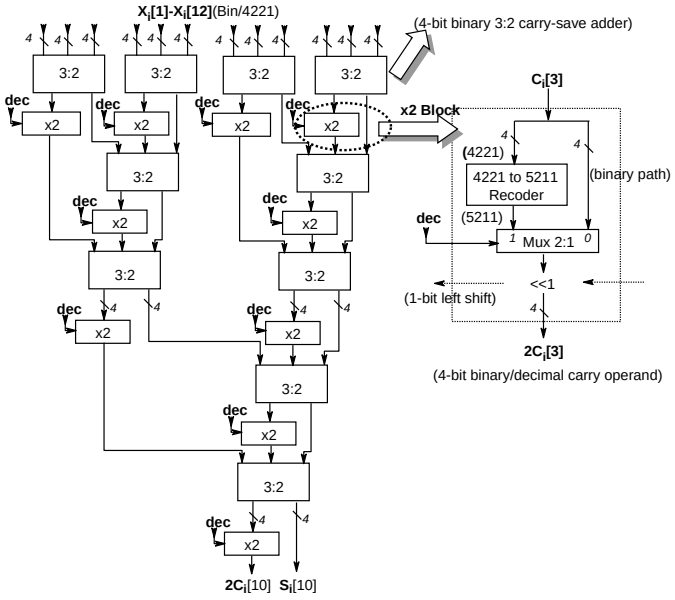

Figure 2: Combined multi-operand adder tree: Basic Implementation (1 digit/4-bit column)

and decimal digit doubling units. A fixed left shift of one bit is denoted by $<<$**1**.

A binary/decimal 3:2 carry-save adder is build in [7] by a straightforward modification of the digit doubling unit of Fig. 1: a 4-bit 2:1 multiplexer is placed after each digit recoder and selects either the carry output of the 3:2 carry-save adder for binary mode or the output of the digit recoder for decimal mode. The output of the multiplexer is shifted one bit to the left. A combined multi-operand adder is implemented as a tree of these modified carry-save adders. Fig. 2 shows an example for 12 operands. The binary/decimal doubling units require an additional input signal **dec** to indicate the operation mode (**dec**=1 for the decimal mode). To reduce the delay and hardware complexity of the combined doubling unit, Hickmann et al. [1], [2] propose to fold the multiplexer into the digit recoding logic. However, since the decimal correction logic is incorporated into the carry-save adder tree, the critical path delay of the binary addition is incremented significantly.

Hickmann et al. [1], [2] propose to improve the latency of the binary operation by splitting the combined carry-save

adder tree into shared, binary and decimal portions. The upper shared portion does not contain any doubling unit in order to avoid penalizing the binary path. However, besides the area penalty due to the replication of parts of the carry-save adder tree, the regular layout of the original binary carry-save tree is practically lost due to this manual redesign.

III. Proposed method

For further area savings a more interesting alternative would be to fully reuse the binary carry-save adder for both binary and decimal multi-operand additions. In this Section we present a method to implement decimal multi-operand addition using any binary carry-save adder (such as a binary 4:2 compressor tree) and separate hardware for decimal correction. The key idea is to perform the decimal doubling $C \times 2$ of the carry operand C coded in 4221 [2] as a 1-bit left shift (C <<1), that is, as a binary doubling. This would allow us to compute both binary and decimal carry-save additions in the same fashion, just by left shifting the carry output of the binary 3:2 carry-save adder.

However, since a left shift of a 4221 decimal coded operand C does not produce exactly its double $2C$, we have to estimate a correction amount to be added to the binary result in order to get the correct decimal sum. Thus, a left shift of a p-digit decimal operand C coded in 4221 produces that each digit C_i is modified as

$$(C_i << 1) = c_{i,3}\ 10 + c_{i,2}\ 4 + c_{i,1}\ 2 + c_{i,0}\ 2$$

On the other hand, the double of a 4221 decimal coded digit is given by

$$C_i \times 2 = c_{i,3}\ (10-2) + c_{i,2}\ 4 + c_{i,1}\ (2+2) + c_{i,0}\ 2$$

The operand $2C = C \times 2$ (represented in code 4221) and the 1-bit shifted operand ($C << 1$) are then related by:

$$2C = (C << 1) + 2 \times \sum_{i=0}^{p-1} (c_{i,1} - c_{i,3})\ 10^i$$

Therefore, we have to increment the decimal correction amount W into +2 units at digit W_i if the carry bit $c_{i,1}$ is one or decrement it by -2 if the carry bit $c_{i,3}$ is one.

For multi-operand addition, each intermediate carry operand $C[k]$ of the binary carry-save adder contributes to the decimal correction amount, but not for the final carry operand, which is multiplied by 2 for decimal. A functional scheme of the proposed method for $4p$-bit binary/p-digit decimal 4221 coded operands is shown in Fig. 3. For simplicity, we consider in Fig. 3 that the m input operands $X[k]$ are aligned to the decimal point and that the sum does not overflow. For m operands, the number of intermediate carry operands $C[k]$ generated in a binary $m:2$ carry-save adder is $m-3$.

The decimal correction amount W is computed in parallel with the binary carry-save addition using an array of bit counters and a decimal carry-save addition. We separate the positive ($c[k]_{i,1}$) and the negative ($c[k]_{i,3}$) carry bits, as soon as

[2] A similar method could be developed for the 5211 code.

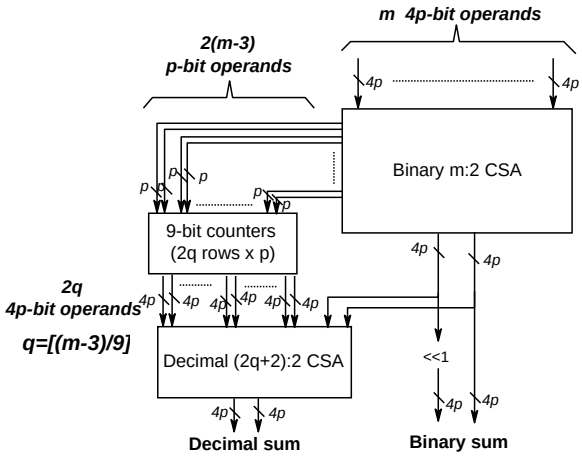

Figure 3: Block diagram of the combined binary/decimal multi-operand addition method

they are generated in the binary carry-save addition, in groups of 9 bits at most for each decimal position i. These groups of bits are added as

$$W_i[2l-1] = \sum_{k=9l-8}^{9l+1} c[k]_{i,3} \quad , \quad W_i[2l] = \sum_{k=9l-8}^{9l+1} c[k]_{i,1}$$

using $2q$ rows of 9-bit counters (or simpler counters) with output coded in 4221, where $q = \lceil (m-3)/9 \rceil$ and l goes from 1 to q. The 4-bit sum value of a 9-bit counter represents a decimal digit $W_i[l] \subset [0,9]$ coded in 4221, so that the output of each row of counters is a decimal operand $W[2l-1]$ or $W[2l]$ of p digits coded in 4221. The decimal correction amount W is given by

$$W = 2 \times \sum_{l=1}^{q} (W[2l] - W[2l-1])$$

Since the representation 4221 is self-complementing [7], the negative operands $-W[2l-1]$ are obtained by a bit inversion of $W[2l-1]$ and a subsequent addition of a unit in the least significant place, that is, as if they were two's complement binary operands. To obtain the final decimal sum $S[2q+m-2]$ and carry $C[2q+m-2] \times 2$ operands, we add the $2q$ operands $W[2l] \times 2$ and $-W[2l-1] \times 2$ to the result of the binary carry-save addition $S[m-2]$, $C[m-2] \times 2$. Since all operands are in 4221 code, we use binary 3:2 carry-save adders and decimal doubling units to perfom a decimal $(2q+2) : 2$ carry-save addition.

IV. Implementation

We consider now some pre-existing binary carry-save adder such a optimized tree of 3:2 or 4:2 compressors and we reuse it to also support decimal multi-operand addition. In Fig. 4 we show a block diagram of 1-digit column (4-bit slice) of the proposed implementation for $m = 12$ input operands. The binary adder tree consists of two levels of 4:2 compressors and one level of 3:2 compressors (binary 3:2 carry-save adder). The 4:2 compressor is build of two levels of 3:2 compressors

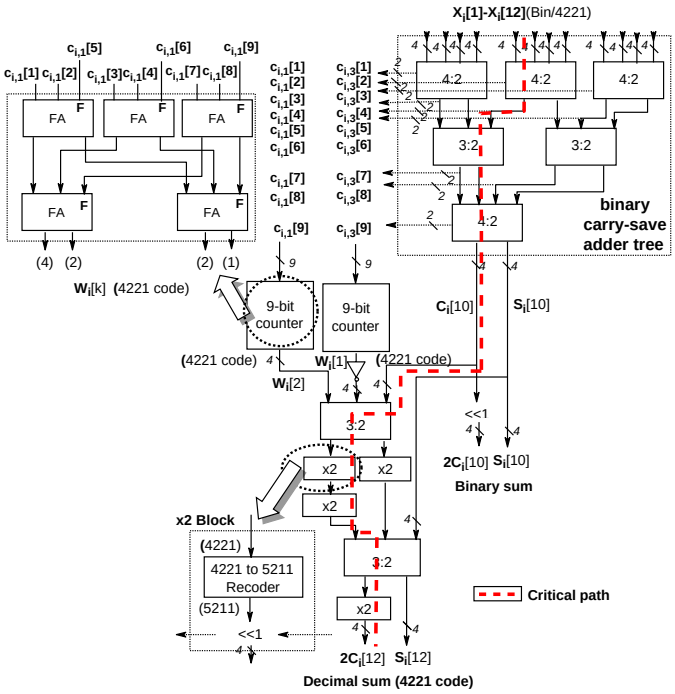

Figure 4: Multi-operand adder tree: Proposed Implementation (1 digit column)

Table II: Size of binary and decimal CSAs for m operands

m	$q = \lceil (m-3)/9 \rceil$	Bin. CSA	Dec. CSA
12	1	12:2	4:2
21	2	21:2	6:2
30	3	30:2	8:2
39	4	39:2	10:2
48	5	48:2	12:2
57	6	57:2	14:2
66	7	66:2	16:2
75	8	75:2	18:2

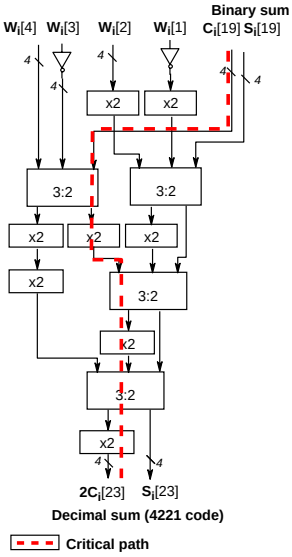

Figure 5: Decimal 6:2 carry-save adder (1 digit column)

optimally interconnected so that the critical path only goes through 3 XOR gates. A total of 9 intermediate carry operands are generated by the compressor tree. The carry bits $c[k]_{i,3}$ and $c[k]_{i,1}$ are summed separately by two 9-bit counters, resulting 4221 coded digits $W_i[1]$ and $W_i[2]$. The internal structure of the 9-bit counter is detailed in the upper left corner of Fig. 4. It is build of 5 full adders arranged in two levels which calculates the sum of the input bits in code 4221. The fastest input goes through 2 XOR levels, while the slowest signals goes through 4 XOR levels. To obtain the negative sum in 4221 code, the counter outputs are inverted. Since carries arrive to the counters with different delays, those produced in the last levels of the binary adder tree are connected to faster inputs of the counter in order to balance the different path delays.

The decimal digits $\overline{W_i[1]}$ and $W_i[2]$, and the sum $S_i[10]$ and carry $C_i[10]$ ouputs of the compressor tree are dispatched to a decimal 4:2 carry-save adder. To reduce the number of doubling units needed, the calculation

$$(\overline{W_i[1]} + W_i[2] + C_i[10]) \times 2 = S_i[11] \times 2 + C_i[11] \times 2 \times 2$$

is performed first using a binary 3:2 carry-save adder and three doubling units. The two cascaded doubling units can be merged into a $\times 4$ unit to obtain a small reduction in area and delay [8]. The critical path of the decimal operation is indicated in Fig. 4 by a thick dotted line. It goes through 15 levels of XOR gates, eight of them corresponding to the binary adder tree. An estimation of its delay is detailed in Section V.

For more than 12 input operands, the decimal carry-save adder required to sum the decimal correction operands $W[l]$ with the output of the binary adder tree gets more complex. In Table II we present the size of the decimal and binary carry-save adders (CSAs) required as a function of m.

The block diagram of a 1-digit decimal 6:2 carry-save adder is shown in Fig. 5. The critical path goes now through 10 XOR gate levels instead of 7. However, as the most part of the computation of the decimal correction is overlapped with the binary carry-save addition, the contribution of the decimal carry-save adder to the total critical path delay remains practically constant as m increases and equal to 10 XOR gate delays. So for higher number m of operands we get better decimal/binary delay ratios.

V. Evaluation and Comparison

To obtain some area and delay estimates, we have modeled the different designs in terms of equivalent XOR gates and XOR delays τ. A full adder has a hardware complexity of 3.5 XOR gates, and a delay of τ (carry and fast input paths) or 2τ. The delay of the 4:2 compressor is 3τ. For the 1-digit recoder we estimate a complexity of 9 XOR gates and a delay of 2τ. Since the building logic elements are the same for the compared tree architectures, we expect to obtain good area and delay ratios for comparison. However, the routing and physical layout complexities of the different proposals cannot be perceived from the result of this comparison.

Table III shows the area and binary/decimal delay estimations for 4-bit/1-digit slice of the proposed binary/decimal

Table III: Area and delay estimations (4-bit/1-digit column)

Combined CSA	Area		Delay	
m:2	(♯ XORs)	% decimal	bin/dec (τ)	Ratio
12:2	235	41%	8/15	1.88
21:2	456	43%	11/21	1.91
30:2	661	42%	11/21	1.91
39:2	866	41%	14/24	1.71
48:2	1071	41%	14/24	1.71
57:2	1276	40%	14/24	1.71
66:2	1481	40%	17/27	1.59
75:2	1686	40%	17/27	1.59

Table IV: Comparison (4-bit/1-digit column CSA trees)

	Area		Delay Bin/Dec	
CSA tree	♯XORs	Ratio	τ	Ratio
64-bit/Decimal64 PPR tree 33:2				
Binary 33:2	434	1	14/–	1/–
Decimal 33:2 [8]	596	1.37	–/22	–/1.57
Proposed	741	1.71	14/24	1.00/1.71
Basic $\times 2$ blocks [7]	668	1.54	26/26	1.86/1.86
Improved $\times 2$ blocks [2]	650	1.50	25/25	1.79/1.79
Split tree [2]	736	1.70	14/25	1.00/1.79
136-bit/Decimal128 PPR tree 69:2				
Binary 69:2	938	1	17/–	1/–
Decimal 69:2 [8]	1280	1.36	–/28	–/1.65
Proposed	1569	1.67	17/27	1.00/1.59
Basic $\times 2$ block [7]	1432	1.53	33/33	1.94/1.94
Improved $\times 2$ block [2]	1394	1.49	32/32	1.88/1.88
Split tree [2]	1616	1.72	17/32	1.00/1.88

multi-operand adders, the percentage of the design area occupied by the the decimal correction hardware and delay ratio of the whole combined unit with respect to the binary carry-save adder tree. The delay of the binary operation is equal to the critical path delay of the binary carry-save adder shown in column 4. A referential decimal carry-save adder tree [8] has around 1.4 times more area and is 1.6 times slower than the equivalent binary carry-save adder. On the other hand, we observe that the decimal correction occupies about 40% of the total area of the design and is between 1.6 to 1.9 times slower than a dedicated decimal adder. Thus, our combined trees offer important area savings of 40% over using separate binary and decimal adder trees while the delay for the decimal operation mode is almost similar to the delay of the referential decimal carry-save adder.

In Table IV, we provide area and delay estimations for different 33:2 and 69:2 binary, decimal, and combined carry-save adder trees already described in previous sections. We have selected values of 33 and 69 for the number of input operands since they correspond to the number of partial products generated by a fixed-point binary/BCD multiplier [2], [7] with support for 64-bit binary/IEEE Decimal64 and 134-bit binary/Decimal128 formats. The basic [7] and improved $\times 2$ block [2] architectures are based on modifications of the doubling units of the decimal tree, which results in an latency penalty of 80% or more over the standalone binary carry-save adder tree. The most attractive architectures for performance are the split tree [2] and ours. Both implementations have very close values of area, and same delays for the binary operation mode as the binary adder tree. The performance improvement comes at expenses of an increase in area of 15% with respect to the improved $\times 2$ block architecture [2]. Overall, our design shows lower delays than the split tree for the decimal operation mode as the number of input operand increases, and a more regular layout and routing. We also expect shorter design times for our implementation due to the possibility of automation design, and reduced power consumption by disabling the decimal hardware on binary mode.

VI. Conclusion and Future Work

A new method for the combined computation of binary/decimal multi-operand additions is presented. It relies on a fully reuse of a binary carry-save adder to reduce area, power consumption and design time. Decimal operands are represented in a 4221 coding different than BCD that allows to perform decimal addition via binary carry-save addition and small decimal corrections. Unlike related proposals, these decimal corrections are computed separately from the carry-save adder tree, with any impact on the latency of the binary operation. Comparison area and delay figures indicate that our architecture presents speed advantages over previous designs with a moderate area overhead. Furthermore, the resulting implementations have regular layouts that make them very appropriate for ultra deep submicron VLSI. Also, it would be relatively simple to incorporate these techniques into a CAD tool for automatic synthesis.

Work in progress includes the synthesis of VHDL register transfer level models of our designs in a 90nm CMOS standard cell library, and the development of combined binary/decimal fixed and floating-point multipliers using the proposed carry-save adder trees for partial product reduction.

Acknowledgment

E. Antelo is supported in part by the Ministry of Education and Science of Spain under contract TIN 2007-67537-C03.

References

[1] M. Erle and B. Hickmann *Combined Binary/Decimal Fixed-Point Multiplier and Method*, US patent, pub. no. 2010/0146030, Jun. 2010.

[2] B. Hickmann, M. Schulte and M. Erle, *Improved Combined Binary/Decimal Fixed-Point Multipliers*, IEEE International Conference on Computer Design (ICCD 2008), pp. 87–94, Oct. 2008.

[3] IEEE Std 754(TM)-2008, *IEEE Standard for Floating-Point Arithmetic*, ISBN 978-0-7381-5753-5, IEEE Computer Society, Aug. 2008.

[4] R. D. Kenney and M. J. Schulte, *High-Speed Multioperand Decimal Adders*, IEEE Transactions on Computers, vol. 54, no. 8, pp. 953–963, Aug. 2005.

[5] 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, vol. 51 , no. 1, pp. 4:1–4:10, Jan/Feb 2009.

[6] C. Tsen, S. Gonzalez-Navarro, M. Schulte, B. Hickmann and K. Compton, *A Combined Decimal and Binary Floating-Point Multiplier*, 20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2009), pp. 8–15, Jul. 2009.

[7] A. Vazquez, E. Antelo, and P. Montuschi, *A New Family of High-Performance Parallel Decimal Multipliers*, 18th IEEE Symposium on Computer Arithmetic, pp. 195–204, Jun. 2007.

[8] A. Vazquez, E. Antelo, and P. Montuschi, *Improved Design of High-Performance Parallel Decimal Multipliers*, IEEE Transactions on Computers, vol. 59, 2010.

[9] A. Vazquez, E. Antelo *VLSI Design of Combined Binary/Decimal Multioperand Adders*, INRIA Research Report, Nov. 2010. Available at http://hal.inria.fr