INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 4, Issue 2 MAY 2015

LOW POWER RADIX-10 MULTIPLICATION FOR DSP/MULTIMEDIA APPLICATIONS

[1]P.Yasodha,
PG Scholar in VLSI System design,
[2]K. Yogitha Bali,
M.Tech, Asst. Professor, ECE Department,
[1]yasodhareddy250@gmail.com,
[2]yogitha.konidala@gmail.com,

ABSTRACT:

This paper introduces two novel architectures for parallel decimal multipliers. Our multipliers are based on a
new algorithm for decimal carry–save multioperand addition that uses a novel BCD–4221 recoding for decimal digits. It significantly improves the area and latency of the partial product reduction tree with respect to previous proposals. We also present three schemes for fast and efficient generation of partial products in parallel. The recoding of the BCD–8421 multiplier operand into minimally redundant signed–digit radix–10, radix–4 and radix–5 representations using new recoders reduces the complexity of partial product generation. In addition, SD radix–4 and radix–5 recodings allow the reuse of a conventional parallel binary radix–4 multiplier to perform combined binary/decimal multiplications. Evaluation results show that the proposed architectures have interesting area–delay figures compared to conventional Booth radix–4 and radix–8 parallel binary multipliers and other representative alternatives for decimal multiplication

Keywords: Parallel multiplication, decimal hardware, overloaded BCD representation, redundant excess-3 code, redundant arithmetic

I. INTRODUCTION

Providing hardware support for decimal arithmetic is becoming a topic of interest. Specifically, the revision of the IEEE–754 Standard for Floating–Point Arithmetic (IEEE–754r) [1] already incorporates specifications for decimal arithmetic. Thus, it is expected that microprocessor manufacturers include decimal floating–point units in their products oriented to mainframe servers to satisfy the high performance demands of current financial, commercial and user–
oriented applications [3].

An important and frequent operation in decimal computations is multiplication. However, due to the inherent in-efficiency of decimal arithmetic implementations in binary logic, practically all the proposed decimal multipliers are sequential units [2, 4, 7, 9, 11, 16]. Recently, the first implementation of a parallel decimal multiplier was presented in [8]. Parallel multipliers are used extensively in most of the binary floating–point units [10, 13] and are of interest for decimal applications to scale performance. In this paper, we introduce new methods for the efficient implementation of decimal parallel multiplication by a parallel generation of partial products and the reductionof these partial products using a novel

decimal carry–save addition tree. We present the architectures of two different high–performance parallel multipliers thatimplement these methods. The second architecture also allows an effective implementation of a combined binary/decimal multiplier. These high–performance implementations have similar hardware complexity or a moderate increment in area with respect to the equivalent binary parallel multipliers.

II. Literature Survey

BCD is a decimal representation of a number directly coded in binary, digit by digit. For example, the number (9321)10 = (1001 0011 0010 0001) BCD.

It can be seen that each digit of the decimal number is coded in binary and then concatenated to form the BCD representation of the decimal number. As any BCD digit lies between [0, 9] or [0000, 1001], multiplying two BCD digits can result in numbers between [0, 81].

All the possible combinations can be represented in a 7-bit binary number when multiplied, (81) 10 or (1010001)2 being the highest. In BCD multiplication where 4-bit binary multipliers are used to multiply two BCD numbers X and Y with digits, Xi and Yj , respectively , a partial product Pij is generated of the form (p6 p5p4p3p2p1p0)2. Conversion of Pij from binary to a BCD number BiCj where π (Xi, Yj) = 10Bi + Cj needs fast and efficient BCD converters.

The binary to BCD conversion is generally inefficient if the binary number is very large. Hence the conversion can be done in parallel for every partial product after each BCD digit is multiplied as shown in Figure 1 and the resulting BCD numbers after conversion can be added using BCD adders.

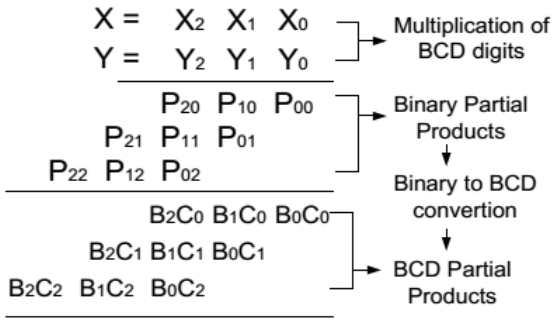

Fig 1: Illustration of BCD conversion in BCD

Multiplication consists of three stages: generation of partial products, fast reduction (addition) of partial products to a two operand and a final carry propagate addition. Decimal multiplication is more complex than binary multiplication mainly for two reasons: the higher range of decimal digits ([0,9]), which increments the number of multiplicand multiples and the inefficiency of representing decimal values in systems based on binary logic using BCD–8421 (since only 9 out of the 16 possible 4–bit combinations represent a valid decimal digit). These issues complicate the generation and reduction of partial products.

The first alternative [2, 4] generates and stores all the required multiplicand multiples. Next, multiples are distributed to the reduction stage through multiplexers controlled by the multiplier digits. This approach requires more than a cycle to generate some complex BCD-8421 multiplicand multiples (3X,6X,7X,8X,9X). To avoid complicated multiples the multiplier can be recoded. In [8] each multiplier digit is recoded as Yi =YH5+YL, with YH$\in${0,1}and YL$\in${−2,−1,0,1,2}. Multiples 2X and 5X can be computed without a carry propagation over the whole number. Negative multiples requires an additional 9's complement addition. The second approach generates only the partial product as needed using digit–by–digit lookup table methods [9, 16]. In a recent work [5], a magnitude range reduction of the operand digits by a radix–10 signed–digit recoding (from [0,9] to [-5,5]) is

HIGH-LEVELARCHITECTURE

suggested. This recoding of both operands speeds–up and simplifies the generation of partial products. Then, overlapped signed–digit partial products 1 are generated using simplified tables and a set of multiplexers and xor gates.

First attempts to improve decimal multiplication performed the reduction of decimal partial products using some scheme for decimal carry propagate addition such as direct decimal addition [12]. Proposals to perform the reduction of decimal partial products using multioperand carry–free addition were suggested in [9] (carry–save) and [15] (signed– digit). Recently several techniques have been proposed that improve these previous works. In [5] a signed–digit decimal adder based on [15] is used. Redundant binary coded decimal (RBCD) adders [14] can also perform decimal carry–free additions using a signed–digit representation of decimal digits ($\in[-7,7]$). In [11] a scheme of two levels of 3–2 binary carry–save adders (CSA) is used to add the partial products iteratively. Since it uses BCD–8421 to represent decimal digits, a digit addition of +6 or +12 (modulo 16) is required to obtain the decimal carry and to correct the sum digit. Logic for detection of decimal carries and sum digit is in the critical path (sum path). In order to eliminate decimal corrections from the critical path of the binary CSA, three different techniques were proposed in [6]. Among these proposals, non–speculative adders present the best area–delay figures and are the most suitable for multioperand addition using a CSA tree. Non–speculative adders reduce the BCD–8421 input operands using a binary CSA tree. Preliminary sum digits are then obtained using a level of 4–bit carry propagate adders. Finally, decimal carry and sum digit corrections are determined from the preliminary sum digit and the carries passed to the next more significant digit position in the binary CSA tree. Decimal correction is performed using combinational logic (its complexity depends on the number of input operands added) and a 3–bit carry propagate adder per digit.

The high-level block diagram of the proposed parallel architecture for dxd-digit BCD decimal integer and fixed-point multiplication is shown in Fig. 1. This architecture accepts conventional (non-redundant) BCD inputs X, Y, generates redundant BCD partial products PP, and computes the BCD product P=XxY. It consists of the following three stages1 : (1) parallel generation of partial products coded in XS-3, including generation of multiplicand multiples and recoding of the multiplier operand, (2) recoding of partial products from XS-3 to the ODDS representation and subsequent reduction, and (3) final conversion to a non-redundant2d-digit BCD product.

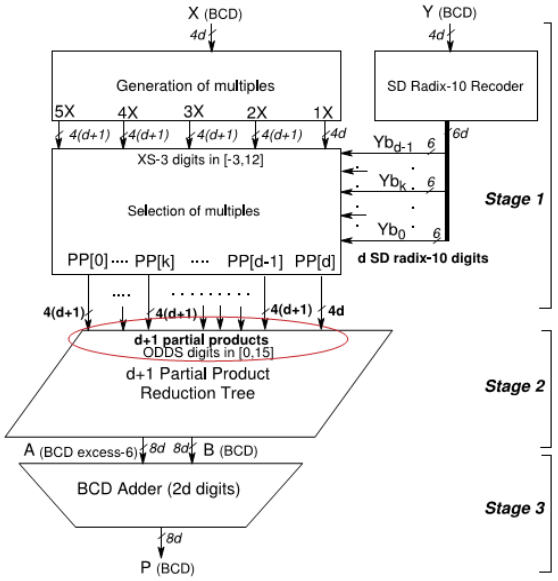

Fig. 2. Combinational SD radix-10 architecture.

Stage 1) Decimal partial product generation:

A SD radix-10 recoding of the BCD multiplier has been used. This recoding produces a reduced number of partial products that leads to a significant reduction in the overall multiplier area. Therefore, the recoding of thed-digit multiplier Y into SD radix-10 digits Ybd-1;...;Yb0,producesdpartial products PP[d-

binary carry-save additions using columns

1]...PP[0],one per digit; note that each Ybk recoded digit is represented in a 6–bit hot-one code to be used as control input of the multiplexers for selecting the proper multiplicand multiple, {-5X;...; -1X;0X;1X;...;5X}. An additional partial product PP[d] is produced by the most significant multiplier digit after the recoding, so that the total number of partial products generated is dþ1

In contrast to our previous SD radix-10 implementations, 3X is obtained in a reduced constant time delay (≈3 XOR-gate delays) by using the XS-3 representation. Moreover, a negative multiple is generated from the correspondent positive one by a bitwise XOR operation. Consequently, the latency is reduced and the hardware implementation is simplified. The scheme proposed also produces 3X in constant time but using redundant signed-digit BCD arithmetic.

Stage 2) Decimal partial product reduction.

In this stage, the array of d+1 ODDS partial products are reduced to two 2d-digit words (A,B). Our proposal relies on a binary carry save adder tree to perform carry-free additions of the decimal partial products. The array ofdþ1ODDS partial products can be viewed as adjacent digit columns of height hdþ1. Since ODDS digits are encoded in binary, the rules for binary arithmetic apply within the digit bounds, and only carries generated between radix-10 digits (4-bit columns) contribute to the decimal correction of the binary sum. That is, if a carry out is produced as a result of a 4-bit (modulo 16) binary addition, the binary sum must be incremented by 6 at the appropriate position to obtain the correct decimal sum (modulo 10 addition). Two previous designs [12], [18] implement tree structures for the addition of ODDS operands. In the nonspeculative BCD adder [18], a combinational logic block is used to determine the sum correction after all the operands have been added in a binary CSA tree, with the maximum number of inputs limited to 19 BCD operands. 2 By contrast, in our method the sum correction is evaluated concurrently with the of binary counters. Basically we count the number of carries per decimal column and then a multiplication by 6 is performed (a correction by 6 for each carry-out from each column).

The result is added as a correction term to the output of the binary carry-save reduction tree.

This improves significantly the latency of the partial product reduction tree. Moreover, the proposed architecture accepts an arbitrary number of ODDS or BCD operand inputs. Some of PPR tree structures presented in [12] (the area-improved PPR tree) also exploit a similar idea, but rely on a custom designed ODDS adder to perform some of the stage reductions.

Our proposal aims to provide an optimal reuse of any binary CSA tree for multioperand decimal addition, as it was one in [31] for the 4221 and 5211 decimal codings.

Stage 3) Conversion to (non-redundant) BCD.

We consider the use of a BCD carry-propagate adder [29] to perform the final conversion to a non-redundant BCD product P=A+B.

The proposed architecture is a 2d-digit hybrid parallel prefix/carry-select adder, the BCD Quaternary Tree adder (see Section 6).

The sum of input digits Ai , Bi at each position i has to be in the range[0,18] so that at most one decimal carry is propagated to the next position i+1.

Decimal partial product generation

The partial product generation stage comprises the recoding of the multiplier to a SD radix-10 representation, the calculation of the multiplicand multiples in XS-3 code and the generation of the ODDS partial products.

PPR tree (h inputs, 1-digit column).

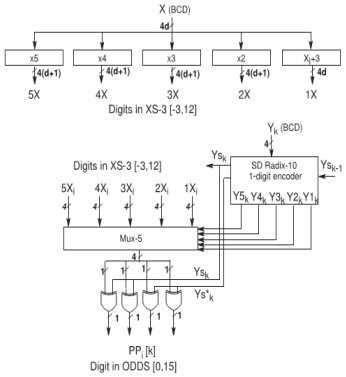

Fig. 3. SD radix-10 generation of a partial product digit

Decimal partial product reduction:

The PPR tree consists of three parts: (1) a regular binary CSA tree to compute an estimation of the decimal partial product sum in a binary carry-save form (S, C), (2) a sum correction block to count the carries generated between the digit columns, and (3) a decimal digit 3:2 compressor which increments the carry-save sum according to the carries count to obtain the final double-word product (A; B), A being represented with excess-6 BCD digits and B being represented with BCD digits. The PPR tree can be viewed as adjacent columns of h ODDS digits each, h being the column height (see Fig. 4), and h≤d+1.

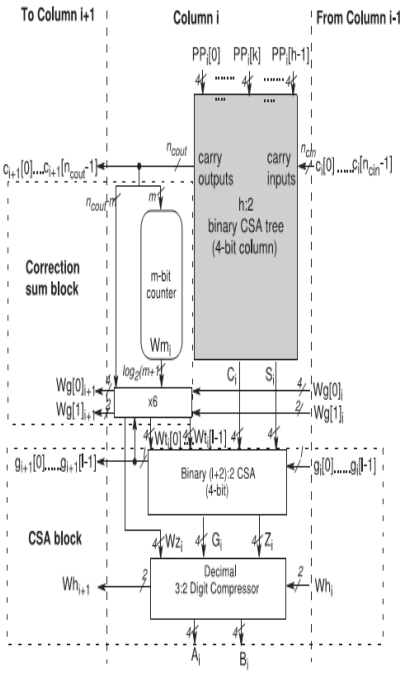

Fig. 5. High-level architecture of the proposed decimal

FINAL CONVERSION TO BCD

The selected architecture is a2d-digit hybrid parallel prefix/ carry-select adder, the BCD Quaternary Tree adder [29]. The delay of this adder is slightly higher to the delay of a binary adder of8dbits with a similar topology. The decimal carries are computed using a carry prefix tree, while two conditional BCD digit sums are computed out of the critical path using 4-bit digit adders which implements [Ai]+Bi+0 and [Ai]+Bi +1. These conditional sums correspond to each one of the carry input values. If the conditional carry out from a digit is one, the digit adder performs a -6 subtraction. The selection of the appropriate conditional BCD digit sums is implemented with a final level of 2:1multiplexers. To design the carry prefix tree we analyzed the signal arrival profile from the PPRT tree, and considered the use of different prefix tree topologies to optimize the area for the minimum delay adder

Conclusion

In this paper we have presented several techniques to implement decimal parallel multiplication in hardware. We propose three different SD encodings for the multiplier that lead to fast parallel and simple generation of partial products. For partial product reduction we have developed a decimal carry–save algorithm based on a BCD–4221 representation of decimal digit operands. It makes possible the construction ofp:2decimal CSA trees that outperform the area–delay figures of existing proposals.

Moreover, proposed techniques also allow the computation of combined binary/decimal multiplications with a moderate overhead. We have proposed an architecture for decimal SD radix–10 parallel multiplication and two combined architectures for binary/decimal SD radix–4 and binary SD radix–4/decimal SD radix–5 multiplication.

The area–delay figures from a comparative study including conventional binary parallel multipliers and other representative decimal proposals show that our decimal SD radix–10 multiplier is an interesting option for high performance with moderate area.

[9] R. H. Larson. High-speed multiply using four input

References

[1] IEEE standard for floating–point arithmetic. IEEE Standards Committee, Oct. 2006.

[2] F. Y. Busaba, T. Slegel, S. Carlough, C. Krygowski, and J. G. Rell. The design of the fixed point unit for the z990 microprocessor.

InProc. ACM Great Lakes14th Symposium on VLSI, pages 364–367, Apr. 2004.

[3] M. F. Cowlishaw. Decimal floating-point: Algorism for computers. InProc. IEEE16th Symposium on Computer Arithmetic, pages 104–111, July 2003.

[4] M. A. Erle and M. J. Schulte. Decimal multiplication via carry-save addition. InProc. IEEE Int'l Conference on Application-Specific Systems, Architectures, and Processors, pages 348–358, June 2003.

[5] M. A. Erle, E. M. Schwarz, and M. J. Schulte. Decimal multiplication with efficient partial product generation. In Proc. IEEE17th Symposium on Computer Arithmetic, pages 21–28, June 2005.

[6] R. D. Kenney and M. J. Schulte. High-speed multioperand decimal adders.IEEE Trans. on Computers, 54(8):953–963, Aug. 2005.

[7] R. D. Kenney, M. J. Schulte, and M. A. Erle. High-frequency decimal multiplier. InProc. IEEE Int'l Conference on Computer Design: VLSI in Computers and Processors, pages 26–29, Oct. 2004.

[8] T. Lang and A. Nannarelli. A radix-10 combinational multiplier. InProc.40th Asilomar Conference on Signals, Systems, and Computers, pages 313–317, Oct. 2006.

carrysave adder. IBM Tech. Disclosure Bulletin, 16(7):2053–2054, Dec. 1973.

[10] N. Ohkubo and M. Suzuki. A 4.4 ns CMOS 54x54–bit multiplier using pass-transistor multiplexer. IEEE Journal of Solid State Circuits, 30(3):251–256, Mar. 1995.