A Variant of a Radix-10 Combinational Multiplier

Luigi Dadda
Politecnico di Milano, Milano, Italy and
Universitá della Svizzera italiana – AlaRI
Lugano, Switzerland
Email: dadda@alari.ch

Alberto Nannarelli
Department of Informatics & Math. Modelling
Technical University of Denmark
Kongens Lyngby, Denmark
Email: an@imm.dtu.dk

Abstract— **We consider the problem of adding the partial products in the combinational decimal multiplier presented by Lang and Nannarelli. In the original paper this addition is done with a tree of decimal carry-save adders. In this paper, we treat the problem using the multi-operand decimal addition previously published by Dadda, where the sum of each column of the partial product array is obtained first in binary form and then converted to decimal. The multiplication, using a 90 nm CMOS technology, in this modified scheme takes 2.51 ns, while in the original scheme it takes 2.65 ns. The area of the two schemes is roughly the same.**

I. Introduction

The need for computing directly decimal numbers (avoiding the decimal-to-binary conversion of input data and the reverse conversion for output data, both necessary with purely binary processors) has been stressed by Cowlishaw [1]. The most relevant data of a future IEEE standard for decimal floating point numbers can be found in [2].

Lang and Nannarelli proposed a combinational 16×16-digit decimal multiplier in [3]. The unit of [3] is organized as follows: the multiplier is recoded in such a way that only multiples 2 and 5 of the multiplicand are required; the partial products are kept in a redundant format; the partial product are accumulated by a tree of redundant adders and the final product is obtained by converting the carry-save tree's outputs into binary-coded decimal (BCD) format. The unit of [3] synthesized in a 90 nm library of standard cells has an operation latency of 2.65 ns and a total area of 300,000μm^2.

In this work, we will deal with the accumulation of partial products by proposing a different architecture for it.

The partial product array of a 16×16-digit multiplier, obtained from the partial product generator of the unit of [3], is shown in Fig. 1. The first row is composed of 17 small discs, each representing a BCD digit. The second row is composed of 17 small circles, each representing a bit. The whole array is composed by 16 pairs of such rows, suitable shifted, as shown in the figure.

In [3] the value of the array is computed via a tree of additions of rows. Each addition is performed using a carry-free decimal adder similar to that proposed by Erle and Schulte [4]. The product is the value of such array, where the weights 10^c of the digits of each column assume the value 10^0 for the rightmost column to 10^{31} for the leftmost column. We compute the value of each decimal column assuming for the binary weights within each column the values 2^3, 2^2, 2^1, 2^0.

In the next sections the basis of the array reduction scheme, implementation details and the results in terms of delay and area are presented. The results show that the variant of the multiplier proposed is about 5% faster and has roughly the same area with respect to the scheme of [3].

II. The carry-save addition of the columns

The column sums can be computed using carry save additions. We show them in dot-notation schemes, as in Fig. 2. Such kind of schemes, introduced in [5] and extended in [6] for obtaining more compact versions, can be easily drawn by hand or, to avoid mistakes, using a program on a spreadsheet [7]. The program for our specific case can be freely downloaded from [8].

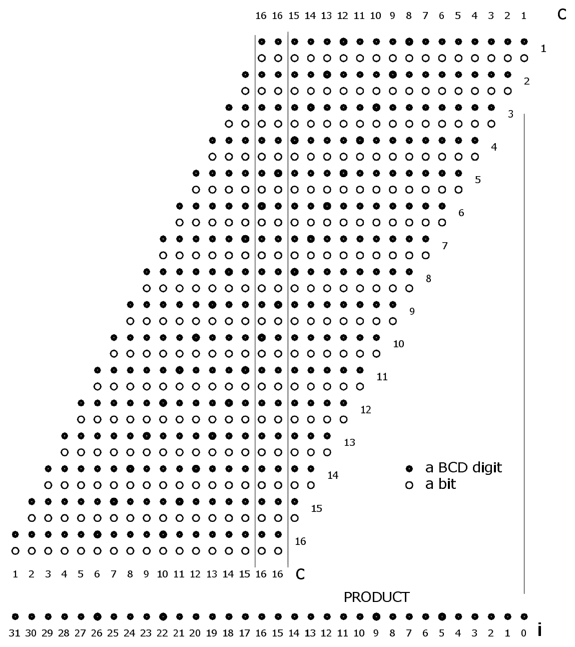

16 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 C
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 16 C
● a BCD digit
○ a bit
PRODUCT
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 i

Fig. 1. The partial product array of a 16×16 digits multiplier of [3].

In Fig. 2, each scheme is marked with an integer c, which is the number of digit-bit pairs composing a column of the array. In the example: $1 \leq c \leq 16$. The first row of each scheme is composed of 3 dots with a number n next to each dot and a fourth dot with a number $2n$. The number n represents the

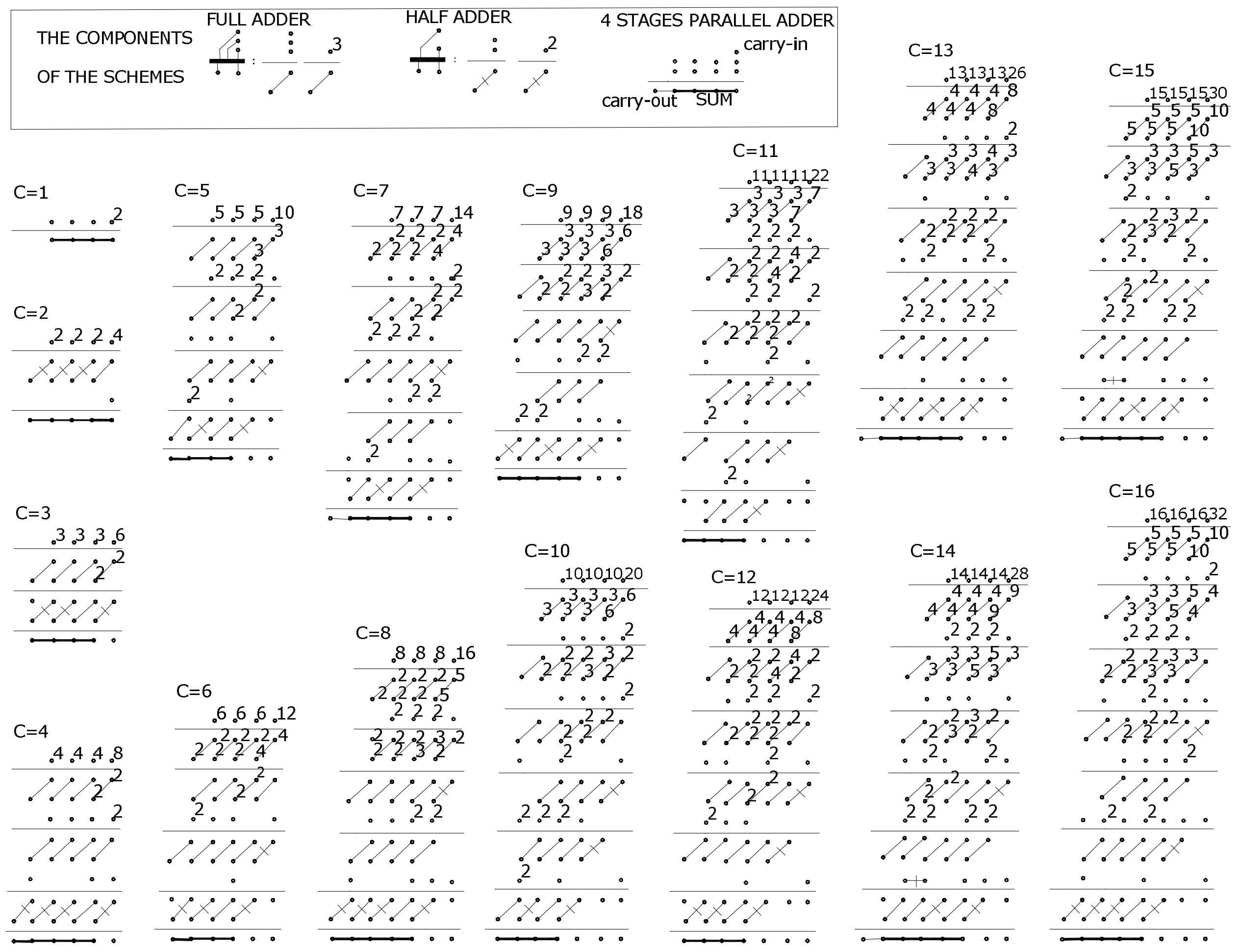


Fig. 2. The binary column adders for a 16×16 digit multiplier.

cardinality of dots having the same weight. A dot with no number implies $n = 1$ and n is omitted for simplicity. The same rule is used for the dots composing each scheme. Each dot scheme (except the one marked $c = 1$) is composed from a number of compression stages each of three rows (except the last composed from two rows). The last row in each scheme is in general composed from a number of dots (in the left part of the row) connected with a thick line, and some isolated dots to its right. The thick line represents a binary adder, the connected dots represent the output variables of such an adder, the inputs (at most two for each output dots) are two variables (dots) per column to be found in the last compression stages composed from two lines only.

The compression algorithm works as follows. Starting from the first row at the top, the number of dots in each binary column is divided by three. The (integer) quotient q represents the number of full adders needed for the compression of the column. They are represented in the three-rows stage that follows by two dots: the *sum* in the same column of the inputs (same weight), and the *carry* in the column at the right of the inputs, plus the number q written next to them. A segment (representing the full adder) joins the two dots. The remainder r of the division ($r = \{0, 1, 2\}$) is represented in the third row with a dot ($r = 1$) or a dot with a 2 ($r = 2$) or without any dot ($r = 0$). Each binary column in a stage has a value given by the sum of the dots (at most three, some can be multiple dots) composing it. The values of the (binary) columns are decreasing through the succeeding stages. When the maximum value of the columns becomes three we use a specific new algorithm for obtaining the successive last stage composed only of columns of at most two dots. This algorithm is very simple: starting from the right side we examine the successive columns: if their value is 1 or 2, we transfer the column to the next stage. When a 3 is found we put in the next stage a full adder. We continue with the column to the left, adding to it the carry just generated. To keep the number of dots in each column not higher than 2, a half adder must be used if a 2 is found in the preceding stage. When a 1 is found in that

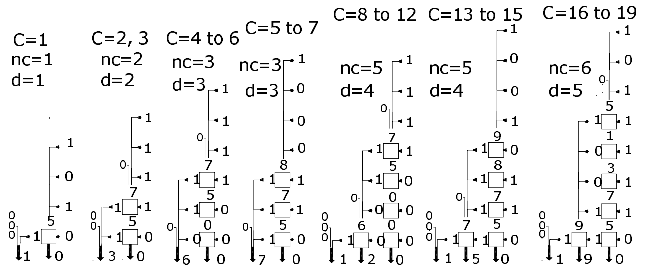


Fig. 3. The BD converters for different values C of digit-bit pairs in a column of Fig. 2 arrays.

stage, we can simply transfer the corresponding dot to the final 2-row stage.

The whole algorithm can be implemented via a spreadsheet as shown in [8]. An interested reader can download it. The program gives also the number of full and half adders, the number of stages, the length in bits of the output of the final binary adder.

In the above carry-free addition, we can also use half adders in the compression stages for obtaining a number of single bits (in the least significant part). This requires a smaller number of stages in the binary parallel adder decreasing both the cost and the total delay. The number of such single bits in the final sum is also given by the spreadsheet program.

III. The Binary-to-Decimal conversion

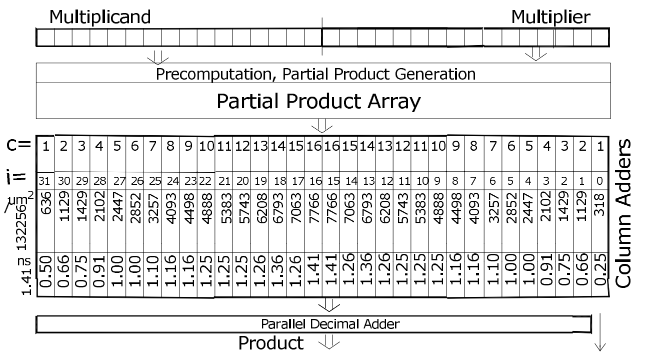
The conversion from binary to decimal has been treated in [9] for the case of adding a number of BCD digits. The same methodology can be applied to the case considered here, of adding a number of BCD digit-bit couples. The conversion schemes use a cell defined by Nicoud [10] (Fig. 3).

All modules in Fig. 3 are composed from a number of identical cells connected in a nearest-neighbor way. Each right inputs and left outputs are binary, while the upper inputs and lower outputs are BCD digits. We now briefly describe the algorithms implemented in a cell.

The (upper) digit input d_i is multiplied by 2 and added to the binary (right) input b_i obtaining $S = 2d_i + b_i$. The maximum value of S is 19; its minimum is obviously 0. We then write the most significant digit of S (i.e. 0 or 1) at its binary output (the left side of the cell); and its least significant digit (0 to 9) at its decimal output (the lower side of the cell).

The decimal input to the topmost cell can either be 3 or 4 bits. If it is 4 bits the corresponding value is 1000 or 1001. In the other case, only the bit of weight 8 in the BCD representation has value 0 and the three bits of weight 1, 2 and 4 determine a value $0, \ldots, 7$ (see Fig. 3 examples).

Note that the binary numbers input to the BD converters correspond to the maximum values expected at the outputs of the various columns, i.e. $10, 20, 30, \ldots, 140, 150, 160$. These values are consequently found at the output of each scheme, at its bottom. Note also that each scheme except the first ($C = 1$) is valid for the range of C shown in the figure.



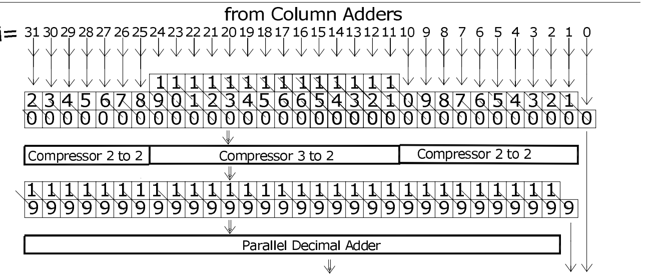


Fig. 4. Top part: the scheme for a 16x16 digit multiplier, with the delay and the area of each column. Bottom part: the maximum values (in skew form) of the outputs of the column adders and of the compressors.

IV. The addition of the Major Partial Products

Fig. 4 shows the scheme of a 16×16 digit decimal multiplier were the Partial Product Array feeds 32 column adders, assumed to include the respective BD converters. The outputs of those converters are shown in a skew-tiled form and compose the Major Partial Product (MPP) array. This appears as a set of tree BCD numbers, the topmost composed by the most significant digits of the column sums, assuming the values 0 or 1 only. The digits composing the second and the third Major Partial Products are generic BCD digits. In order to obtain the sum of the three MPPs we first compress each column into an equivalent set of two digits, through a compressor whose dot-scheme is shown in Fig. 5.

Compressors in dot-schemes have been introduced in [11], [12]. An extension to the decimal case has been shown in [9] with a family of compressors applicable to decimal columns with any number of digits, the decimal carry-free addition [4] being the simplest case.

The scheme of Fig. 5 shows an input of two BCD digits, each represented by 4 dots, and a single bits in the rightmost place, that feeds the carry input of the first stage.

The scheme is composed from a 4-stage binary adder (a carry-look-ahead adder, for speed reason), and a single BD conversion cell. The cell decimal output represents the digit d_0 having the same decimal weight of the input digits, while the single bit from the binary output represents the digit with the weight of the column input to the left of the column input to the compressor.

Note that the columns from $i = 11$ to $i = 24$ (see Fig. 4)

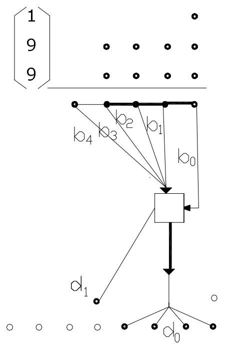


Fig. 5. A decimal carry-save adder, or column compressor.

component	delay $[ps]$	area $[\mu m^2]$
full-adder	100	90
half-adder	50	45
BD	50	118
CLA-4	200	200
CLA-5	210	350
CLA-6	220	415
PPG	700	155K
CPA	400	10K

TABLE I
DELAY AND AREA FOR COMPONENTS IN MULTIPLIER.

can have a 1 in the first row of the MPP array. The column from $i = 1$ to $i = 10$ and from $i = 25$ to $i = 31$ (last) use the same compressor, since it is desirable to have as inputs to the final Decimal Adder one of the addends composed of 0s or 1s only. The decimal adder is in this case somewhat simpler as shown in [3].

V. Time and area

From the scheme of Fig. 4, it is possible to compute the multiplier delay (in particular the maximum delay, in the critical path) and the total silicon area.

Such data are computed here with the assumption that a 90 nm technology is used for the implementation. All the data of the basic components have been derived by synthesis, using Synopsys Design Compiler and the STM library of standard cells.

The basic components are full and half-adders, Binary-to-Decimal conversion cells (BD), and 4, 5, 6 bit carry-look-ahead binary adders (CLA). The components' values of delay and area are listed in Table I. In addition, for the delay and area of the partial product generation (PPG) and of the final decimal adder (CPA) we use the value presented in [3] (listed in the two bottom lines of Table I).

By using the spreadsheet program of [8], we can compute the delay and area of each column in the array. Those values are reported in the top part of Fig. 4.

Consequently, the multiplier with the array reduction scheme of Fig. 4, has a total delay of 2.51 ns, and the total area of 297,256μm^2.

With respect to the implementation of [3], the operation latency is 5% shorter and the area slightly smaller (less than 1%).

Moreover, such spreadsheet programs allow an immediate recalculation in case of variations of the data on the basic components; as an example, we can consider the case of adopting implementations optimized for the area[1] (at 90 nm technology), or with a different technology (e.g 65 nm technology).

[1]The components of Table I are optimized for minimum delay.

It seems of some interest to mention a previous multiplier designed by Dadda on a different principle, namely the calculation of the binary value of the n^2 digit products [7]. This multiplier obtains the same total delay (2.51 ns) with an area of 347,000μm^2.

VI. Conclusion

It has been shown how the partial products addition in the decimal multiplier proposed in [3], based on a tree of decimal carry save adders, can be obtained with trees (one per column) of binary carry save adders followed by binary-to-decimal converters.

The total delay is reduced from 2.65 ns to 2.51 ns (with a 90 nm technology) while the the area is 297,256μm^2 in this scheme and 300,000μm^2 in the scheme of [3].

References

[1] M. F. Cowlishaw, "Decimal floating-point: algorism for computers," in *Proc. of 16th Symposium on Computer Arithmetic*, June 2003, pp. 104–111.

[2] IBM. Decimal Arithmetic Specification . Feb. 2003. [Online]. Available: http://www2.hursley.ibm.com/decimal/decbits.html

[3] T. Lang and A. Nannarelli, "A Radix-10 Combinational Multiplier," *Proc. of 40th Asilomar Conference on Signals, Systems, and Computers*, pp. 313–317, Nov. 2006.

[4] M. Erle and M. Schulte, "Decimal Multiplication via Carry-save Addition," in *Proc. of 14th International Conference on Application-Specific Systems, Architectures and Processors*, July 2003, pp. 337–347.

[5] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 19, pp. 349–356, Mar. 1965.

[6] ——, "A compact dot notation for the design of binary adders, multipliers and adders of products," *AlaRI internal report*, Dec. 2005.

[7] ——. Spreadsheet tools for the design of a parallel decimal multiplier. AlaRI internal report, Dec. 2005. [Online]. Available: http://www.alari.ch/people/dadda/

[8] ——. Spreadsheet tools for the design of a radix-10 combinational multiplier. AlaRI internal report, 2007. [Online]. Available: http://www.alari.ch/people/dadda/

[9] ——, "Multi Operand Parallel Decimal Adders: a mixed Binary and BCD Approach," *IEEE Transactions on Computers*, vol. 56, pp. 1320–1328, Oct. 2007.

[10] J. Nicoud, "Iterative Arrays for Radix Conversion," *IEEE Transactions on Computers*, vol. C-20, pp. 1479–1489, Nov. 1971.

[11] L. Dadda, "On parallel multipliers," *Alta Frequenza*, vol. 45, pp. 574–580, 1976.

[12] D. Gajski, "Parallel Compressors," *IEEE Transactions on Computers*, vol. C-29, pp. 1479–1489, May 1980.