Binary-coded decimal digit multipliers

G. Jaberipur and A. Kaivani

Abstract: With the growing popularity of decimal computer arithmetic in scientific, commercial, financial and Internet-based applications, hardware realisation of decimal arithmetic algorithms is gaining more importance. Hardware decimal arithmetic units now serve as an integral part of some recently commercialised general purpose processors, where complex decimal arithmetic operations, such as multiplication, have been realised by rather slow iterative hardware algorithms. However, with the rapid advances in very large scale integration (VLSI) technology, semi- and fully parallel hardware decimal multiplication units are expected to evolve soon. The dominant representation for decimal digits is the binary-coded decimal (BCD) encoding. The BCD-digit multiplier can serve as the key building block of a decimal multiplier, irrespective of the degree of parallelism. A BCD-digit multiplier produces a two-BCD digit product from two input BCD digits. We provide a novel design for the latter, showing some advantages in BCD multiplier implementations.

1 Introduction

Decimal computer arithmetic is preferred in decimal data processing environments such as scientific, commercial, financial and Internet-based applications [1]. Ever growing needs for processing power, required by applications with intensive decimal arithmetic, cannot be met by conventional slow software simulated decimal arithmetic units [1]. However, their hardware counterparts as an integral part of recently commercialised general purpose processors [2] are gaining importance. Binary-coded decimal (BCD) encoding of decimal digits has conventionally dominated decimal arithmetic algorithms, whether realised by hardware or in software.

The research for hardware realisation of decimal arithmetic is not matured yet and there are rooms for improvements in hardware algorithms and designs. For example, the state-of-the-art BCD multipliers, for computing $X \times Y$, use iterative multiplication algorithms [3, 4], where the partial products (i.e. the product of one BCD digit of the multiplier Y times the multi-BCD-digit multiplicand X) are generated one at a time and added to the previously accumulated result. Each partial product may be directly generated as one BCD number in $[0, 9] \times X$, or may be composed of few easy multiples of the multiplicand (e.g. $7X = 4X + 2X + X$) [5]. The latter approach tends to increase the depth (measured by the maximum number of equally weighted BCD digits) of partial product tree per each BCD digit of multiplier, which in general leads to slower partial product accumulation. But, by using possibly fast and low-cost BCD digit by BCD-digit multipliers, the former approach may lead to less costly BCD multipliers.

Erle *et al.* have enumerated three reasons for using decimal digit-by-digit multipliers for partial product generation, which leads to less number of cycles, less wiring and no need for registers to store multiples of the multiplicand [4]. With the rapid advances in VLSI technology, semi(fully)-parallel BCD multipliers will soon be attractive, where more than one (all) partial product(s) are generated at once and accumulated in parallel. An integral building block of a BCD multiplier, whether realising a sequential, semi- or fully parallel multiplication algorithm, can be the BCD-digit multiplier. Alternative approaches are based on either slow accumulation of easy multiples [5], or costly retrieval of product of BCD digits from look-up tables [6, 7].

2 General BCD multiplication

A general conventional paper and pencil view of decimal multiplication is depicted in Fig. 1, where in this figure and throughout the paper uppercase (lowercase) letters are used for decimal (binary) digits. Each decimal digit product $X_i \times Y_j$ is represented by two decimal digits P_{ij}^h and P_{ij}^l such that the former weighs ten times as much as the latter; hence, h (for high) and l (for low) superscripts. Using BCD encoding for all decimal digits of Fig. 1 leads to a general BCD multiplication scheme, for which a hardware implementation may be achieved by one of the following sequential, semi- or fully parallel approaches.

2.1 Sequential realisation

The product is generated in a register initialised to zero. In each iteration the multi-digit multiplicand is multiplied by one decimal digit of the multiplier, and the resulted partial product is accumulated in the product register, followed by a digit right shift. Depending on the partial product generation approach, to be discussed later, there may be equally weighted BCD digits in the representation of a single partial product (e.g. P_{01}^h and P_{02}^l and similar pairs in Fig. 1 rise to two deep partial products). Therefore the accumulation step may actually be equivalent to a multi-operand BCD addition. Fig. 2 depicts an abstract exemplary hardware realisation, where the three-operand BCD addition box receives a two-deep partial product and the one-deep accumulated result.

doi:10.1049/iet-cdt:20060160

Paper first received 19th September and in revised form 25th December 2006

The authors are with the Department of Electrical and Computer Engineering, Shahid Beheshti University, School of Computer Science, Institute for Studies in Theoretical Physics and Mathematics (IPM), Tehran 19839-63113, Iran

E-mail: jaberipur@sbu.ac.ir

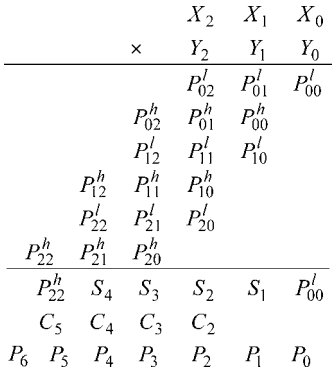

Fig. 1 *Three-digit BCD multiplication*

2.3 Semi-parallel realisation

This is similar to the previous one, except that in each iteration, more than one digit of multiplier takes part in partial product generation, which leads to a deeper multi-operand BCD addition. A two digit at a time realisation with a five-operand addition is depicted in Fig. 3.

2.4 Fully parallel realisation

Here, all partial products are generated at once and reduced together to two partial products to be added by a BCD adder. This case may be illustrated as in Fig. 1.

The problem of multi-operand BCD addition, as needed in the realisation of partial product reduction/accumulation, is generally discussed in [8]. But, for BCD partial product generation, one can think of two approaches:

2.5 BCD digit multiplication

This follows the conventional paper and pencil approach, but a two BCD digit product may be looked up in a table addressed by the bits of two BCD digits of the multiplier and multiplicand [7], or a direct BCD digit by BCD-digit multiplier may be realised.

2.6 Precomputed easy multiples of the multiplicand

A straightforward approach is to generate all the ten possible multiples of the multiplicand at the outset of multiplication process. Then, in each iteration, a ten-way selector controlled by a BCD digit of the multiplier selects the appropriate partial product and adds it to the so far accumulated result.

To reduce the number of precomputed multiples, to save multiple generation and selection hardware, a clever design

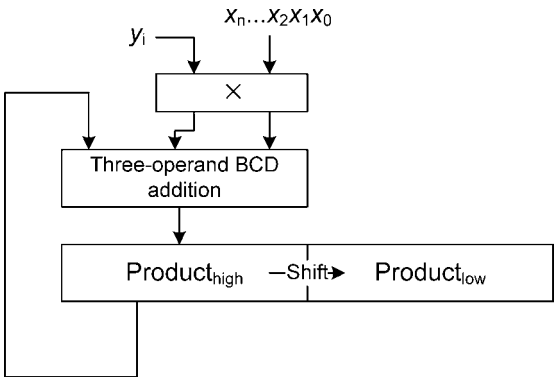

Fig. 2 *Sequential BCD multiplication*

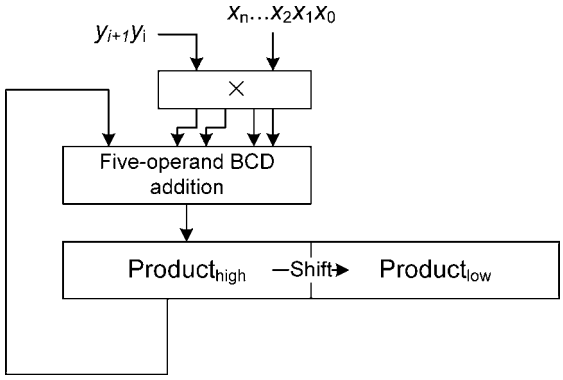

Fig. 3 *Semi-parallel BCD multiplication*

is presented in [3], where only two, four and five multiples of multiplicand are precomputed. With these easy multiples and the multiplicand itself, all the required ten multiples can be derived by at most one carry-free BCD addition without necessarily using any redundant BCD representation. The clever observation, leading to selection of the latter three multiples, is that each BCD digit of the multiplicand when multiplied by 2, 4 or 5, results in a pair of decimal carry and a BCD digit (c, $W = w_3w_2w_1w_0$), such that for multiple 2, $c \leq 1$ and $w_0 = 0$; for multiple 4, $c \leq 3$ and $w_1 = w_0 = 0$, and for multiple 5, $c \leq 4$ and W is either 0 or 5. The latter characteristic guarantees that addition of carries to the equally weighted BCD digits will not generate any further carries.

3 BCD digit multiplication

The BCD encoding of decimal digits [0, 9] maps the latter set to [0000, 1001] such that $x_3x_2x_1x_0$ as the BCD encoding of X $(0 \leq X \leq 9)$ satisfies the arithmetic equation $8x_3 + 4x_2 + 2x_1 + x_0 = X$. A BCD-digit multiplier, with two BCD digits X and Y, realises a function $\pi(X, Y) = X \times Y$, returning a product value in [0, 81] represented by two BCD digits B and C, such that $\pi(X, Y) = 10B + C$, where $0 \leq B\ (C) \leq 8\ (9)$. The function π may be realised, in a straightforward manner, by an eight-input, eight-output combinational logic or a 256×8 look-up table. But practical constraints on area and latency call for more optimum designs.

An alternative design may use a standard 4×4 unsigned binary multiplier generating an 8-bit binary output, which should be corrected to two BCD digits, with the same arithmetic value. Given that the product value belongs to [0, 81], its most significant bit (weighted 2^7) is always zero. Let $X = x_3x_2x_1x_0$ and $Y = y_3y_2y_1y_0$ represent the two input BCD digits and $p_6p_5p_4p_3p_2p_1p_0$ is the output (i.e. product) of the standard 4×4 multiplier, with $p_7 = 0$ ignored. Fig. 4 depicts the regular partial product generation and reduction process of this multiplier.

In binary parallel multiplication, there are several techniques for partial product reduction (e.g. [9, 10]) and final product computation. For wide word operands (e.g.

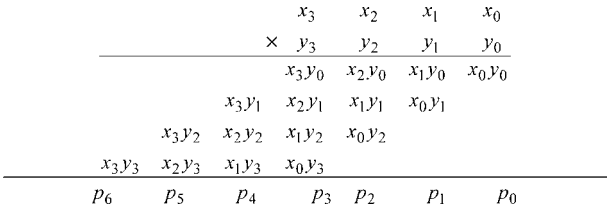

Fig. 4 *BCD × BCD → Binary*

80	40	20	10	4	3	2	1
0	p_6	p_5	p_4	0	p_2	p_1	p_0
0	0	p_6	p_5	0	p_4	p_4	0
				0	p_6	p_5	0
				p_3	0	0	0
b_3	b_2	b_1	b_0	c_3	c_2	c_1	c_0

Fig. 5 *Binary product to BCD conversion: the principle*

popular 54 × 54 bit multipliers [11]), the latter techniques show considerable efficiency. But in decimal multiplication, because of particularities of using radix 10, which is not a power of 2, one needs to generate BCD partial products to be followed by BCD multi-operand addition. Therefore we need localised reduction trees, as in Fig. 4, per each BCD-digit multiplication and alternative customised reduction techniques for better performance, to be discussed in Section 4. But, in this section, we proceed with converting the binary product $p_6p_5p_4p_3p_2p_1p_0$ to its equivalent BCD product $B\ C$, as depicted in Fig. 5. Although the general binary-to-BCD conversion is extensively addressed in the literature (e.g. [12–14]), we have managed to design a special, simpler and faster, binary-to-BCD converter as depicted in Fig. 6.

The first row in Fig. 5 shows the BCD weights. The weights of p_3, p_2, p_1 and p_0 are the same as the corresponding weights in the original binary number $p_6p_5p_4p_3p_2p_1p_0$. But weights 16, 32 and 64 of p_4, p_5 and p_6, have been decomposed to (10, 4, 2), (20, 10, 2) and (40, 20, 4), respectively. p_3 has been moved to fourth row to avoid the possibility of violating the interval [0, 9] for each row-filled BCD digit. The four BCD digits in the right four columns of Fig. 5 may be added, by a BCD adder, to lead to the BCD digit C ($c_3c_2c_1c_0$) of the product $B\ C$ and a decimal carry to be added to the two BCD digits in the left three columns of Fig. 5 leading to B ($b_3b_2b_1b_0$). Fig. 6 depicts the required circuitry, where its correctness has been checked through VHDL (Very high speed integrated circuit Hardware Description Language) simulation, and black-filled boxes show the critical delay path. Here, we only note that because $0 \leq p_6p_5p_4p_3p_2p_1p_0 \leq 81$, the following logical hold

$$p_6p_5 = 0 \qquad p_6p_4(p_3 + p_2 + p_1) = 0 \tag{1}$$

4 BCD partial product reduction

In BCD encoding of decimal digits, bit strings 1010 to 1111 are not used. This leads to some bit interdependencies, which may be beneficiary in designing a simpler and faster partial product tree for BCD digit multiplication.

x_3y_3	$x_3y_2 + x_2y_3$	$x_3y_1 + x_2y_2 + x_1y_3$	$x_3y_0 + x_2y_1$ $x_1y_2 + x_0y_3$	x_2y_0 x_0y_2 x_1y_1 $x_1y_1x_0y_0$	x_1y_0
p_6	p_5	p_4	p_3	p_2	

Fig. 7 *Compact partial product tree*

x_3y_3	$x_3y_2 + x_2y_3$	$x_3y_1 + x_2y_2 + x_1y_3$ $x_2y_2x_1y_1x_0y_0$	$x_3y_0 + x_2y_1$ $x_1y_2 + x_0y_3$ c	$x_2y_0 \oplus x_0y_2 \oplus x_1y_1$

Fig. 8 *Further reduced partial product tree*

Definition 1: (BCD constraint): Given that a BCD digit $X = x_3x_2x_1x_0$, because $0 \leq X \leq 9$, does not assume all the 16 possible bit strings, the constraints $x_3x_2 = 0$ and $x_3x_1 = 0$ hold. □

Using the latter constraint on the bits of both X and Y, the partial product tree of Fig. 4 may be redrawn as the one in Fig. 7, where + is used to indicate a logical OR operation. Note that the items in the tree of Fig. 7 have been produced by adding the items in the relevant columns of Fig. 4, using the BCD constraint for simplifications.

Summation of the four operands in the third column from right (i.e. position of p_2) may produce a carry for position of p_4 or a carry to position of p_3, respectively, represented as $x_2y_2x_1y_1x_0y_0$ and c in Fig. 8, where c is easily derived as

$$c = x_0y_0(x_1y_1 \oplus x_2y_2) + \overline{x_0y_0}x_1y_1(x_2y_2 \oplus x_0y_2) \tag{2}$$

To compute the binary product $p_6p_5p_4p_3p_2p_1p_0$, we use a carry look-ahead logic to add the items in positions p_3 to p_5, as depicted in Fig. 9. It turns out that because of BCD constraint, defined above, no carry passes through position of p_5. The overall delay of the circuits of Figs. 6 and 9, when cascaded, amounts to ten logic levels, where the black-filled gates show the critical path.

In iterative multiplication, often the system-determined iteration cycle-time allows for more latent BCD digit multipliers. Therefore one may focus on area optimisation. The logic of Fig. 10 also depicts a binary product BCD digit multiplier, but with more delay and less area compared to that of Fig. 9. Note that although p_6 is the most latent output of the circuit in Fig. 10, the critical delay path of the whole multiplier, realised by cascading the circuit of Fig. 6 at the output of the circuit of Fig. 10, goes through p_5 and the overall delay amounts to that of 13 logic levels. We will show, in the next section, that iterative BCD multipliers of the previous works can easily accommodate the latency of our area-optimised BCD digit multiplier.

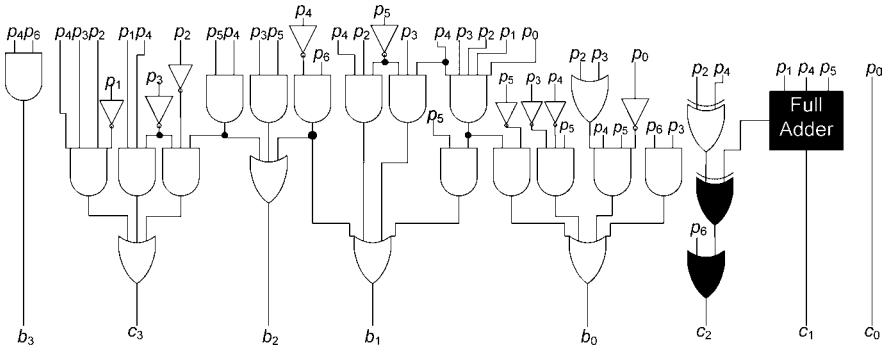

Fig. 6 *Binary product to BCD conversion: the logic*

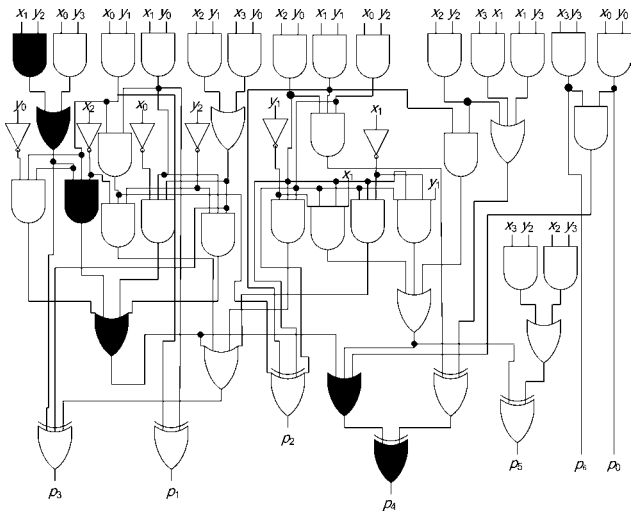

Fig. 9 *Delay-optimised binary product BCD digit multiplier*

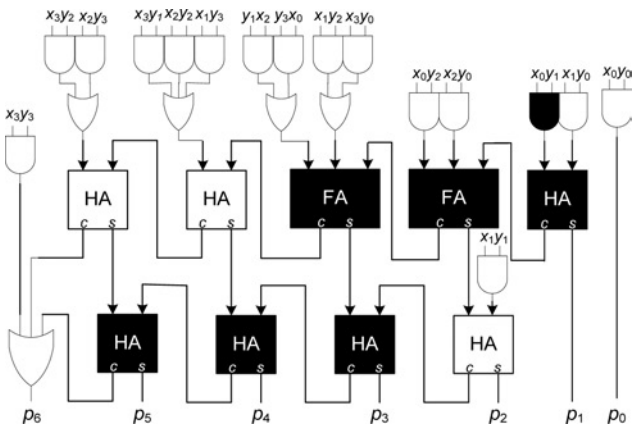

Fig. 10 *Area-optimised binary product BCD digit multiplier, FA (HA): full (half) adder*

5 Comparison with previous works

We have not encountered any direct implementation for BCD digit multipliers in the literature, except for look-up table implementations (e.g. [6, 7]). The latest work based on decimal digit-by-digit multiplier converts the BCD operands to signed digits in $[-5, 5]$ and uses a signed-digit-by-signed-digit multiplier on a word-by-digit basis to generate the partial products, also represented by signed digits [4]. The latter work does not provide any area and time measures that can be used as a comparison basis.

To compare our results with other published works, we have designed iterative BCD multipliers based on the delay-optimised and area-optimised BCD-digit multipliers of the previous section. One hardware realisation of BCD multipliers [3] uses the iterative approach with precomputed easy multiples as explained in Section 2. Our approach for partial product generation is different from that of [3], but both designs use the same method for partial product accumulation. Therefore for the sake of accurate comparison, we deemed it enough to run simulations only on the first part of multipliers, and measured areas of the partial product generation logic for the two approaches through simulation based on a 0.25 μm Complementary metal oxide semiconductor (CMOS) standard process. We had to use our own version of equations for the five multiples because of seemingly wrong equations in [3]. For further explanations on this claim see the appendix. It turns out that the area of our delay-optimised and area-optimised designs is 13% and 30% less than that of [3], respectively.

The partial product generation based on the easy multiples is faster than our partial product generation scheme with a latency of 13 logic levels, as derived in Section 4. But in a pipeline design this is not a disadvantage, for the partial product generation takes up one stage of the pipeline whose cycle-time is determined by the latency of the most latent pipeline stage, which happens to be the partial product reduction stage with 13 logic levels as explained below. The iterative BCD multiplier of [3] uses a special (4:2) compressor for partial product reduction. The latter function is realised by a seven-logic-level BCD digit adder (implemented based on the design in [15]) followed by a six-logic-level simplified one, where the second operand is a single bit.

More regularity in VLSI implementation may be considered as another advantage of our approach. The reason lies in using only one cell (i.e. BCD-digit multiplier) in the whole partial product generation logic. But in the easy multiples method, different cells for different multiples and $4n$-bit four-way multipliers are used.

Another iterative multiplier [16] uses redundant decimal digits for representation of intermediate partial products. It operates in 14% higher clock frequency than that of [3] and ours, but requires 77% more area than that of [3], and certainly much more than ours.

6 Conclusion

We have designed a novel BCD-digit multiplier cell that can be used in conventional iterative BCD multiplier circuits. We showed that this design alternative leads to 30% savings in the area of partial product generation logic. It does neither affect the rest of the multiplier circuitry, nor does it add to the overall delay of a pipelined implementation. Our design leads to more regular VLSI implementation, and does not require special registers for storing easy multiples. Further research is on going on efficient use of the designed BCD-digit multiplier in semi- and fully parallel BCD multipliers.

7 Acknowledgment

The authors wish to thank the unanimous reviewers for their valuable comments. This research was supported, in part, by Shahid Beheshti University under Grant no. 185/1175, and also in part by IPM under Grant no. CS1385-3-02.

8 References

1 Cowlishaw, M.F.: 'Decimal floating-point: algorism for computers'. Proc. 16th IEEE Symposium on Computer Arithmetic, June 2003, pp. 104–111
2 Busaba, F.Y., Krygowski, C.A., Li, W.H., Schwarz, E.M., and Carlough, S.R.: 'The IBM Z900 decimal arithmetic unit'. Asilomar Conf. on Signals, Systems and Computers, November 2001, vol. 2, pp. 1335–1339
3 Erle, M.A., and Schulte, M.J.: 'Decimal multiplication via carry-save addition'. Conf. on Application-Specific Systems, Architectures, and Processors, June 2003, pp. 348–358
4 Erle, M.A., Schwartz, E.M., and Schulte, M.J.: 'Decimal multiplication with efficient partial product generation'. 17th IEEE Symp. on Computer Arithmetic, (ARITH-17), June 2005, pp. 21–28
5 Ohtsuki, T., Oshima, Y., Ishikawa, S., Yabe, K., and Fukuta, M.: 'Apparatus for decimal multiplication'. U.S. Patent 4677583, June 1987

6 Ueda, T.: 'Decimal multiplying assembly and multiply module'. U.S. Patent 5379245, January 1995
7 Larson, R.H.'High-speed multiply using four input carry save adder', *IBM Technical Disclosure Bull.*, 1973, **16**, (7), pp. 2053–2054
8 Kenney, R.D., and Schulte, M.J.: 'High-speed multioperand decimal adders', *IEEE Trans. Comput.*, 2005, **54**, (8), pp. 953–963
9 Wallace, C.S.: 'A suggestion for fast multiplier', *IEEE Trans. Electron. Comput.*, 1964, **13**, pp. 14–17
10 Dadda, L.: 'Some schemes for parallel multipliers', *Alta Frequenza*, 1965, **34**, pp. 349–356
11 Goto, G., Sato, T., Nakajima, M., and Sukemura, T.: 'A 54 × 54-b regularly structured tree multiplier', *IEEE J. Solid-State Circuits*, 1992, **27**, (9), pp. 1229–1236
12 Schmookler, M.: 'High-speed binary-to-decimal conversion', *IEEE Trans. Comput.*, 1968, **17**, (5), pp. 506–508
13 Rhyne, V.T.: 'Serial binary-to-decimal and decimal-to-binary conversion', *IEEE Trans. Comput.*, 1970, **19**, (9), pp. 808–812
14 Arazi, B., and Naccache, D.: 'Binary-to-decimal conversion based on the $2^8 - 1$ by 5', *Electron. Lett.*, 1992, **28**, (23), pp. 2151–2152
15 Schmookler, M., and Weinberger, A.: 'High-speed decimal addition', *IEEE Trans. Comput.*, 1971, **20**, (8), pp. 862–866
16 Kenney, R.D., Schulte, M.J., and Erle, M.A.: 'A high-frequency decimal multiplier'. IEEE Int. Conf. Computer Design: VLSI in Computers and Processors (ICCD), Oct 2004, pp. 26–29

9 Appendix

The equations provided in [3], for computing $5 \times X$, seem to be faulty. For example, try $5 \times 1 = (0101) \times (0001)$, which leads to 1 (0001), where the correct result is obviously 5 (0101).

The correct set of equations may be derived as follows

Let $X = X_{n-1} \ldots X_i X_{i-1} \ldots X_1 X_0$, and $Y = 5 \times X = (X_{n-1} \ldots X_i X_{i-1} \ldots X_1 X_0 0)/2$, where each BCD digit X_i $(0 \leq i \leq n-1)$ is represented by $x_i^3 x_i^2 x_i^1 x_i^0$. We divide each BCD digit of X by 2. Then $Y_i = (X_{i-1} - x_{i-1}^0)/2 + (10 \times x_i^0)/2 = (0 x_{i-1}^3 x_{i-1}^2 x_{i-1}^1) + 0 x_i^0 0 x_i^0$. The latter binary addition of two BCD digits, as explained in the end of Section 2, does not generate any decimal carry, and leads to the following equations

$$y_i^0 = x_i^0 \oplus x_{i-1}^1, \quad y_i^1 = x_{i-1}^2 \oplus x_{i-1}^1 x_i^0,$$

$$y_i^2 = x_{i-1}^3 \oplus x_i^0 \overline{x_{i-1}^2 x_{i-1}^1}, \quad y_i^3 = x_i^0 (x_{i-1}^3 + x_{i-1}^2 x_{i-1}^1)$$