High-Speed Parallel Decimal Multiplication with Redundant Internal Encodings

Liu Han, *Student Member*, *IEEE*, and Seok-Bum Ko, *Senior Member*, *IEEE*

Abstract—The decimal multiplication is one of the most important decimal arithmetic operations which have a growing demand in the area of commercial, financial, and scientific computing. In this paper, we propose a parallel decimal multiplication algorithm with three components, which are a partial product generation, a partial product reduction, and a final digit-set conversion. First, a redundant number system is applied to recode not only the multiplier, but also multiples of the multiplicand in signed-digit (SD) numbers. Furthermore, we present a multioperand SD addition algorithm to reduce the partial product array. Finally, a digit-set conversion algorithm with a hybrid prefix network to decrease the number of the logic gates on the critical path is discussed. An analysis of the timing delay and an HDL model synthesized under 90 nm technology show that by considering the tradeoff of designs among three components, the overall delay of the proposed 16×16-digit multiplier takes about 11 percent less timing delay with 2 percent less area compared to the current fastest design.

Index Terms—Decimal arithmetic, parallel multiplication, redundant number system, multioperand SD adder, hybrid prefix network

1 Introduction

Decimal computer arithmetic is becoming more attractive in the financial and commercial computing area which includes currency conversion, billing and banking system, and tax calculation, since its binary counterpart has an inherent defect in aforementioned applications [1]. Due to the advantage of decimal computer arithmetic in financial and commercial applications, a specification is added into the IEEE standard for floating point arithmetic in IEEE 754-2008 [2]. In the meantime, hardware solutions which support decimal computation are released by IBM [3], [4], [5] and SilMinds [32]. Software libraries are supported by Intel [6], ANSI C [33], and GCC [34]. A survey of hardware designs for decimal arithmetic is provided by Wang et al. in [7].

Multiplication is one of the four basic arithmetic operations. An analysis of benchmarks shows that the percentage of execution time of decimal multiplication could reach over 27 percent in some applications [8]. Due to the importance of multiplication, some decimal fixed-point designs are proposed in [9], [10], [11], [12], [13], [15], [16], [17], [25]. Furthermore, decimal floating-point multipliers based on those fixed-point designs are published in [19], [20], [21], [30], [31].

This paper presents a parallel decimal multiplication algorithm which recodes the multiples into a specific redundant digit-set to reduce the number of the partial products and simplify the partial product reduction (PPR). In the meantime, the carry propagation penalty in the partial product generation (PPG) is avoided. Moreover, in order to reduce partial products, the multioperand SD addition algorithm which reduces SD partial products by binary arithmetic units and generates the decimal transfer digit and interim sum by combinational recoders is discussed. Finally, in our proposed design, a digit-set conversion algorithm to obtain the result in binary coded decimal (BCD) encoding is introduced. Novel aspects of the proposed decimal multiplication algorithm are listed as follows:

1. A PPG algorithm which creates lower number of bits in partial products and single-bit carry for each partial product without carry propagation,
2. A PPR method based on the decimal multioperand SD addition algorithm, and
3. An optimized digit-set conversion algorithm with a hybrid carry prefix network.

Throughout this paper, two input operands and one output product are represented by X, Y, and R, respectively. The encoding of the signal is indicated at the subscript. For a signal S, the index of a digit in it is indicated by the subscript S_i, and the index of a bit inside one digit is indicated by the superscript S_i^j. For example, the third bit of the fourth digit of a signal X is represented as X_3^2. Note that all indexes start from 0. The digit width of the input is n. Thus, the product in BCD encoding is in $2n$ digits, and the bit width of the internal signal is expressed by the multiple of n. $\overline{S}$ represents the signed digit with a negative sign of the value of S. The curly brackets concatenate the signals in between them.

The rest of this paper is organized as follows: Section 2 is a brief review of the current available related works. In Section 3, we provide an overview of the proposed multiplication. The detailed algorithm and structure of the PPG, the PPR, and the final digit-set conversion are provided in Sections 4, 5, and 6, respectively. A comparison among the proposed design and referred algorithms is provided and discussed in Section 7. Finally, Section 8 concludes the paper.

• *The authors are with the Department of Electrical and Computer Engineering, University of Saskatchewan, 57 Campus Dr., Saskatoon, SK S7N 5A9, Canada. E-mail: {liu.han, seokbum.ko}@usask.ca.*

Manuscript received 14 June 2011; revised 14 Nov. 2011; accepted 16 Jan. 2012; published online 23 Jan. 2012.
Recommended for acceptance by B. Parhami.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-06-0395.
Digital Object Identifier no. 10.1109/TC.2012.35.

2 Previous Work

Erle and Schulte propose a traditional method of decimal multiplication in [9]. The design borrows the idea from binary multiplication which reduces the partial products in a carry save adder (CSA)-based structure. Furthermore, to reduce the complexity of the multiples generation, a so-called secondary set which contains $\{X, 2X, 3X, 4X, 8X\}$ is applied, and all the missing multiples could be generated based on the elements in the secondary set with no more than one carry save addition. The decimal 3:2 CSA and 4:2 compressor are described in this paper. Furthermore, the partial product for each iteration could be added iteratively within the delay of a decimal 4:2 compressor. An $n \times n$-digit multiplication can be finished in $n + 4$ cycles.

A sequential redundant multiplication is described in [10]. The authors present an algorithm which recodes both operands into the SD digit-set $[-5, 5]$ to generate the SD operands with simple logic. Further, a digit multiplier block on the range of $[2, 5] \times [2, 5]$ is proposed to generate the partial products in SD format. Hence, a Svoboda signed digit adder with a restricted range is consequently applied to add signed digit partial products iteratively. The SD sequential multiplier takes $n + 4$ cycles to finish one multiplication.

An alternative sequential decimal multiplier in which the secondary set is represented as $\{X, 2X, 4X, 5X\}$ is proposed in [11]. Additionally, a 2-stage overloaded decimal adder which can sum two partial products and one iteration result with less delay than a decimal 4:2 compressor is presented. By doing so, a clean-up block has to be applied to finally correct the decimal encoding before the carry propagated addition in the final step. Thus, in such a multiplier, the latency of one operation is up to $n + 8$ cycles.

In [12], to avoid complicated multiples of X, the operand Y is recoded into two parts, $Y_i = Y_{Hi} + Y_{Li}$, where $Y_H \in \{0, 5, 10\}$ and $Y_L \in \{-2, -1, 0, 1, 2\}$. Therefore, only the $-2X, -X, 2X$, and $5X$ need to be implemented in logic gates. Since the multiples are represented in 10's complement format, the negation is implemented by a 9's complement recoder, and the incremental one is only applied on the least significant digit (LSD). Furthermore, to generate the partial products from $1X$ to $9X$ in BCD carry save (BCD-CS) format, a decimal CSA has to be applied. The parallel PPR for $2n$ partial products (i.e., n sums and n carries) is implemented by six levels of BCD full adders (BCD-FA) for a 16×16-digit multiplication. Half of the decimal carries of partial products are added separately by carry counters. Two outputs of PPR, $2n$-digit sum and $2n$-bit carry, are added together by a prefix network with a conditional adder. Furthermore, an improved PPR algorithm based on a multioperand decimal addition in [14] is provided by Dadda and Nannarelli in [13]. The partial products in columns are first added in a binary form with the binary carry save adder. Subsequently, a binary to decimal conversion algorithm is applied to convert the binary result to decimal encoding.

In [15], Jaberipur and Kaivani propose a new PPG algorithm which only generates $2X$ and $5X$ to compose other multiples from $1X$ to $7X$. The $8X$ and $9X$ are divided into two parts in which the $8X$ is implemented by $E = 10E_h + E_l$, and the $9X$ is implemented in the same way as $N = 10N_h + N_l$. Therefore, the algorithm avoids not only the negation logic for $-2X$ and $-X$, but also the $4X$ (double times $2X$) to generate $8X$ and $9X$. Furthermore, by analyzing the range of the computation and gate level representation, the BCD-FA in the PPR unit is simplified. The two outputs of the PPR unit are further reduced to one $2n$ digits and one $(2n - l)$-digit BCD numbers, where l is the number of levels of the BCD-FA in the PPR structure. Hence, for the final product computation, only $2n - l$ digits are involved in the carry propagation adder to generate the final multiplication result.

In [25], the authors propose a redundant decimal addition algorithm based on a specific encoding, namely weighted bit-set (WBS) encoding. With such an addition algorithm, a multiplier based on the redundant number system is provided. The double-BCD format multiples are first created by combining the easy decimal multiples (i.e., $2X$, $4X$, and $5X$). In the PPR unit, two-operand redundant adder is applied to reduce $2n$ BCD partial products to a redundant number in the range of $[0, 15]$, so called overloaded decimal digit set (ODDS). Furthermore, in the last step, the redundant product is converted to the BCD encoding by a digit set converter with a propagation process.

In [16], Vázquez et al. propose an improved design of their previous work published in [23]. In the improved new family parallel decimal multiplier, two unconventional decimal encodings (i.e., BCD-4221 and BCD-5211) and two architectures (i.e., radix-10 and radix-5) are applied to generate and reduce the partial product. In radix-10 architecture, the operand Y is recoded into SD digit-set $[-5, 5]$, and $n + 1$ partial products are selected by the recoded Y. Alternatively, in radix-5 architecture, the second operand is encoded into two parts, $Y_i = Y_i^U \cdot 5 + Y_i^L$, where $Y_i^U \in \{0, 1, 2\}$ and $Y_i^L \in \{-2, -1, 0, 1, 2\}$. Therefore, in this scheme, there are $2n$ partial products need to be reduced. In the PPR unit, only the binary full adders (BFA) and combinational recoders are applied due to the specific encodings. Finally, two $2n$-digit results are added together with a quaternary tree (Q-T) adder based on the conditional speculative decimal addition proposed by the same authors in [22].

3 Proposed Multiplication Scheme

In the proposed design, we recode one of the two operands into the digit-set $[-5, 5]$, and represent the multiples of the other operand from $-5X$ to $5X$ in the digit-set $[-8, 8]$. By doing so, all the multiples could be obtained in a constant delay, and only $n + 1$ partial products, namely PP, are generated. Furthermore, to reduce the $n + 1$ levels of partial products into the final SD result, a multilevel multioperand SD addition is discussed. To reduce the delay and area of the hardware in PPR unit, we apply binary arithmetic units and combinational recoders in the multioperand SD adder. Finally, a digit-set converter with hybrid carry propagation network is applied to convert the product from SD to BCD encoding. In the proposed hybrid prefix tree, we apply different prefix trees with less digit width to construct a big prefix carry propagation network. Consequently, in the prefix tree, the levels of prefix nodes after the longest column in the PPR unit are reduced. Overall, the structures of the PPG, PPR, and final converter are balanced, and the delay of the proposed multiplier is optimized. The top level architecture of the proposed multiplication is shown in Fig. 1.

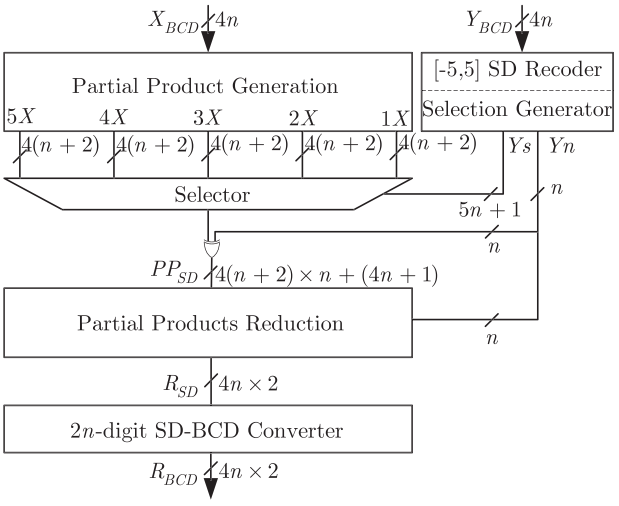


Fig. 1. Top-level architecture of the proposed parallel decimal multiplication.

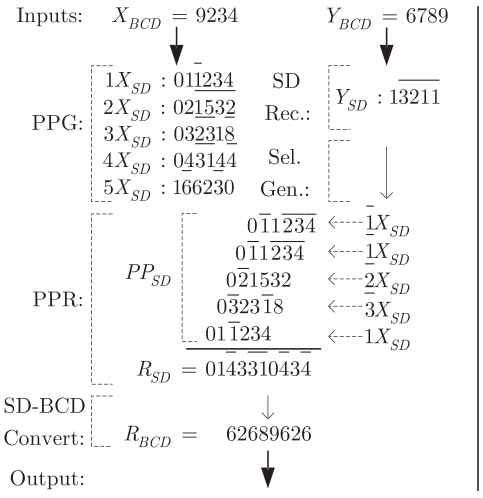


Fig. 2. Example of the proposed 4×4-digit multiplication algorithm.

In the first stage, the n-digit operand Y_{BCD} is recoded into $(n+1)$-digit Y_{SD} in digit-set $[-5, 5]$. The 5-bit "hot one" selection signal Ys_i for each digit is generated based on the recoded operand, Y_{SD}. In the proposed design, only the positive multiples, X through $5X$, are implemented by logic gates. The negative multiples, $-X$, $-2X$, $-3X$, $-4X$, and $-5X$, could be represented in the similar way. However, to reduce the area of the multiplier, we generate the negative multiples by inverting the sign on each digit of the positive multiples. Since the digit in the proposed SD multiples is represented in 2's complement encoding, the inversion is done by an XOR gate controlled by Yn_i which is also the increment bit for each digit to invert the sign of the multiples. Note that only one bit is enough to invert a partial product; therefore, the increment bits for all digits in a partial product are identical.

The second stage in the proposed multiplication is a PPR unit implemented by multiple levels of multioperand SD adders. For example, in a 16×16-digit multiplication, after the PPG unit, there are 17 partial products need to be reduced. We rearrange the layout of the partial product array to apply two levels of SD adders to generate the final product in SD format. In such an multioperand SD addition, the operands could be first reduced by the binary arithmetic unit, and in the end, a recoder is applied to correct the transfer digit and interim sum within the decimal manner. Thus, in the proposed multiplication, the decimal correction is compacted as much as possible.

In the third stage, to convert the SD product back to BCD encoding efficiently, a hybrid carry propagation network which consists of several small carry prefix networks is provided to counterbalance the different delays on different bits of the result of the PPR unit. Compared to traditional methods, the hybrid prefix tree has less level and more level of nodes on the middle and less significant digits of the result from the PPR unit, respectively. Since the middle columns in a partial product array consume more delay than the ending columns, the overall delay of the multiplier could be further reduced with the proposed hybrid prefix tree. An example to illustrate each component of a proposed 4×4-digit multiplier is provided in Fig. 2.

4 SD Partial Product Generation

In the proposed multiplier, we follow the SD radix-10 method described in [16] to recode Y_{BCD} into the SD digit-set $[-5, 5]$. To represent multiples, we provide a new method which generates $n+1$ partial products without carry propagation. In Table 1, the positive multiples from $1X$ to $5X$ are

TABLE 1
Signed Digit Representation of the Proposed Multiples

BCD Operand	$1X_i$		$2X_i$		$3X_i$		$4X_i$		$5X_i$			Y_i	
	T_{i+1}	W_i	T_{i+1}	W_i	T_{i+1}	W_i	T_{i+1}	W_i	K_{i+2}	T_{i+1}	W_i	T_{i+1}	W_i
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	2	0	3	1	-6	0	0	5	0	1
2	0	2	1	-6	1	-4	1	-2	0	1	0	0	2
3	0	3	1	-4	1	-1	1	2	0	1	5	0	3
4	0	4	1	-2	2	-8	2	-4	1	-8	0	0	4
5	0	5	1	0	2	-5	2	0	1	-8	5	1	-5
6	1	-4	1	2	2	-2	3	-6	1	-7	0	1	-4
7	1	-3	1	4	2	1	3	-2	1	-7	5	1	-3
8	1	-2	2	-4	3	-6	3	2	1	-6	0	1	-2
9	1	-1	2	-2	3	-3	4	-4	1	-6	5	1	-1
$T_i + W_i$	$[-4, 6]$		$[-6, 6]$		$[-8, 6]$		$[-6, 6]$					$[-5, 5]$	
$K_i + T_i + W_i$									$[-8, 7]$				

represented in the SD digit set $[-8, 7]$. Thus, a 4-bit 2's complement number could be applied to represent each digit in the multiples from $1X$ to $5X$. To reduce the area of the PPG unit, the negative multiples are generated by inverting the sign on each digit of the corresponding positive multiples, and the digit-set for all multiples from $-5X$ to $5X$ is extended to $[-8, 8]$ (i.e., $\{[-8, 7] \cup [-7, 8]\} \in [-8, 8]$). Unlike the binary signed digit encoding proposed in the decimal signed digit number system in [29], to invert the sign of each digit, an increment bit is involved in the proposed encoding system. Therefore, one signed digit in the proposed multiples is represented by 5 bits. However, the penalty on the hardware area is minimized, since the increment bits for all digits in a partial product are identical.

4.1 Multiples Generation

In Table 1, all the multiples could be divided into two parts except the $5X_i$ which is divided into three parts. To simplify the representation of the multiplies generation, we define three variables in Table 1, where W_i represents the residual number which has the same weight as the current BCD digit. T_{i+1} and K_{i+2} are the transfer digits to the next two digits which have 10 and 100 times weight of the current BCD digit, respectively. The sum of the three variables is restricted in the range of $[-8, 7]$ to form one digit in SD number. Since the variables can be directly generated according to different inputs, and the carry (transfer digit) never propagates exceeding three neighbor digits, the delay of the proposed PPG is independent on the width of the operand. In addition, for an n-digit operand, each multiple contains $n + 2$ SD digits. The SD multiple could be obtained by adding K_i, T_i, and W_i with a 4-bit adder after a recoder generating these variables. Due to the specific converting pattern in Table 1, the conversion can be assumed as a constant addition. Thus, we optimize the 4-bit add operation and convert it to the combinational logic to reduce area and delay. The equations of one digit of the positive multiples are listed below. Note that the signals on the right side of the equal sign is in BCD encoding, and the signals on the left side of the equal sign is in proposed SD encoding.

1X: Since the digit-set $[-8, 7]$ is applied to generate the positive multiples in the proposed PPG algorithm, $1X$ has to be converted to the target digit-set. In the equation, the signal T_i represents the incoming transfer digit which is determined by the previous one digit

$$\begin{aligned}
T_i &= X_{i-1}^3 + X_{i-1}^2 X_{i-1}^1 \\
1X_i^3 &= X_i^3\left(\overline{X}_i^0 + \overline{T}_i\right) + X_i^2 X_i^1 \\
1X_i^2 &= T_i X_i^1 X_i^0 + X_i^3\left(\overline{X}_i^0 + \overline{T}_i\right) + X_i^2 \\
1X_i^1 &= T_i X_i^0\left(\overline{X}_i^3 \overline{X}_i^1 + X_i^2\right) \\
&\quad + \left(\overline{X}_i^0 + \overline{T}_i\right)\left(\overline{X}_i^2 X_i^1 + X_i^3\right) \\
1X_i^0 &= T_i \oplus X_i^0.
\end{aligned} \tag{1}$$

2X: In the proposed algorithm, since the transfer digit from last digit in multiple $2X$ could be from 0 to 2, two bits (i.e., T_i^1 and T_i^0) are needed to represent the incoming transfer digit

$$\begin{aligned}
T_i^1 &= X_{i-1}^3 \\
T_i^0 &= X_{i-1}^2 + X_{i-1}^1 \\
2X_i^3 &= \overline{X}_i^0\left(\overline{T}_i^1 X_i^2 \overline{X}_i^1 + X_i^3\right) + \overline{X}_i^2 X_i^1 + \overline{T}_i^1 X_i^3 \\
2X_i^2 &= X_i^0\left(T_i^1 \overline{X}_i^3 \overline{X}_i^2 + X_i^1\right) + T_i^1 X_i^1 \\
&\quad + \overline{X}_i^0\left(\overline{T}_i^1 X_i^2 \overline{X}_i^1 + X_i^3\right) + \overline{T}_i^1 X_i^3 \\
2X_i^1 &= \overline{X}_i^2 \overline{X}_i^1\left(T_i^1 \overline{X}_i^0 + \overline{T}_i^1 X_i^0\right) \\
&\quad + \left(\overline{T}_i^1 \overline{X}_i^0 + T_i^1 X_i^0\right)\left(X_i^1 + X_i^2\right) \\
2X_i^0 &= T_i^0.
\end{aligned} \tag{2}$$

5X: To generate multiple $5X$ in digit-set $[-8, 7]$, two transfer digits which have 10 and 100 times weight of the current digit are needed. Since only two elements are in the digit-sets of the residual number W_i and the transfer digit K_i, the logic could be simplified as shown in

$$\begin{aligned}
W_i &= X_i^0 \\
K_i &= X_{i-2}^3 + X_{i-2}^2 \\
5X_i^3 &= X_{i-1}^3(\overline{K}_i + \overline{W}_i) + X_{i-1}^2 \\
5X_i^2 &= W_i\left(\overline{X}_{i-1}^3 + \overline{K}_i\right) \\
5X_i^1 &= W_i K_i \overline{X}_{i-1}^3 + X_{i-1}^3(\overline{K}_i + \overline{W}_i) \\
&\quad + X_{i-1}^1(K_i + W_i) \\
5X_i^0 &= X_{i-1}^1 \oplus (W_i \oplus K_i).
\end{aligned} \tag{3}$$

3X: By applying the redundant number system to represent the partial product, the $3X$ logic does not contain the carry propagation in digit level any more. Thus, a constant delay in PPG could be achieved

$$\begin{aligned}
T_i^1 &= X_{i-1}^3 + X_{i-1}^2 \\
T_i^0 &= X_{i-1}^3 + \overline{X}_{i-1}^2 X_{i-1}^1 \\
3X_i^3 &= X_i^3\left(\overline{X}_i^0 + \overline{T}_i^0 + \overline{T}_i^1\right) + X_i^2 \overline{X}_i^1 \\
&\quad + X_i^1\left(\overline{T}_i^0 \overline{T}_i^1 \overline{X}_i^2 + \overline{X}_i^0\left(\overline{X}_i^2 + \overline{T}_i^1\right)\right) \\
3X_i^2 &= T_i^1 X_i^3 \overline{X}_i^0 + X_i^1\left(\overline{T}_i^0 \overline{T}_i^1 \overline{X}_i^2 + \overline{X}_i^0\left(\overline{X}_i^2 \overline{T}_i^1\right)\right) \\
&\quad + X_i^0\left(T_i^0 T_i^1 X_i^2 + \overline{X}_i^1\left(T_i^0 \overline{X}_i^3 + T_i^1 \overline{T}_i^0\right) + \overline{T}_i^1 X_i^3\right) \\
3X_i^1 &= \overline{X}_i^3\left(\overline{T}_i^1 \overline{T}_i^0 X_i^0 + T_i^1\left(\overline{X}_i^0 + T_i^0\right)\right)\left(\overline{X}_i^1 + \overline{X}_i^2\right) \\
&\quad + \left(T_i^1 \overline{T}_i^0 X_i^0 + \overline{T}_i^1\left(\overline{X}_i^0 + T_i^0\right)\right)\left(X_i^2 X_i^1 + X_i^3\right) \\
3X_i^0 &= T_i^0 \overline{X}_i^0 + \overline{T}_i^0 X_i^0.
\end{aligned} \tag{4}$$

4X: The multiple $4X$ in the proposed work is not generated based on two times of $2X$ as in other works. A direct and simple method is shown below

$$
\begin{aligned}
4X_i^3 &= \overline{X}_i^3\overline{X}_i^2\overline{X}_i^1X_i^0 + X_i^2X_i^1\overline{X}_i^0 \\
&\quad + \overline{X}_{i-1}^0\big(\overline{X}_i^2\overline{X}_i^1X_i^0 + X_i^2\overline{X}_i^0\big) \\
&\quad + \overline{X}_{i-1}^3\big(X_i^1\overline{X}_{i-1}^2\big(\overline{X}_i^0 + X_i^2\big) + \overline{X}_i^2\overline{X}_i^1X_i^0 + X_i^2\overline{X}_i^0\big) \\
4X_i^2 &= X_i^3\big(X_{i-1}^3\overline{X}_{i-1}^0 + X_{i-1}^2\big) \\
&\quad + X_i^0\big(\overline{X}_i^3X_{i-1}^3\big(\overline{X}_i^1X_{i-1}^0 + \overline{X}_i^2\big) \\
&\quad + \overline{X}_{i-1}^3\big(X_i^2X_i^1\overline{X}_{i-1}^2 + X_i^3\big) + \overline{X}_i^2X_{i-1}^2\big) \\
&\quad + \overline{X}_i^0\big(\overline{X}_i^2\big(\overline{X}_i^1X_{i-1}^3X_{i-1}^0 + X_i^1\overline{X}_{i-1}^3\overline{X}_{i-1}^2\big) \\
&\quad + X_i^2\big(X_{i-1}^3\big(\overline{X}_{i-1}^0 + X_i^1\big) + \overline{X}_i^1\overline{X}_{i-1}^3 + X_{i-1}^2\big)\big) \\
4X_i^1 &= \overline{X}_{i-1}^2\big(X_{i-1}^0 + \overline{X}_{i-1}^3\big)\big(X_i^3\overline{X}_i^0 + \overline{X}_i^3\overline{X}_i^2X_i^0 + X_i^1\big) \\
&\quad + \big(X_{i-1}^3\overline{X}_{i-1}^0 + X_{i-1}^2\big)\big(\overline{X}_i^1\big(\overline{X}_i^3\overline{X}_i^0 + X_i^2\big) + X_i^3X_i^0\big) \\
4X_i^0 &= \overline{X}_{i-1}^3\overline{X}_{i-1}^2X_{i-1}^0 + X_{i-1}^3\overline{X}_{i-1}^0 + X_{i-1}^1.
\end{aligned} \tag{5}
$$

4.2 Partial Product Selection

In the proposed multiplier, a minimally redundant radix-10 digit-set $[-5, 5]$ is applied to represent the operand Y. Since the recoded set is symmetrical, and the multiples are encoded in signed digit number, the selection signals for the negative multiples are the same as the positive multiples (i.e., $Ys_i^{4\ldots0}$ indicate the signals to select $\pm5X, \ldots, \pm1X$). If a negative multiple is selected, a one-bit negation signal Yn_i for each selected partial product is applied to invert the signs of all digits in the corresponding positive multiple. The equations for the selection signal and negation signal are given in

$$
\begin{aligned}
T_i &= Y_{i-1}^2\big(Y_{i-1}^0 + Y_{i-1}^1\big) + Y_{i-1}^3 \\
Ys_i^4 &= \overline{Y}_i^2\overline{Y}_i^1\big(T_i\overline{Y}_i^0 + \overline{T}_iY_i^0\big) \\
Ys_i^3 &= T_iY_i^0\big(\overline{Y}_i^3\overline{Y}_i^2\overline{Y}_i^1 + Y_i^2Y_i^1\big) \\
&\quad + \overline{T}_i\overline{Y}_i^0\big(\overline{Y}_i^2Y_i^1 + Y_i^3\big) \\
Ys_i^2 &= Y_i^1\big(T_i\overline{Y}_i^0 + \overline{T}_iY_i^0\big) \\
Ys_i^1 &= T_iY_i^0\big(Y_i^2\overline{Y}_i^1 + \overline{Y}_i^2Y_i^1\big) + \overline{T}_iY_i^2\overline{Y}_i^0 \\
Ys_i^0 &= Y_i^2\overline{Y}_i^1\big(T_i\overline{Y}_i^0 + \overline{T}_iY_i^0\big) \\
Yn_i &= Y_i^3\big(\overline{Y}_i^0 + \overline{T}_i\big) + Y_i^2\big(Y_i^0 + Y_i^1\big).
\end{aligned} \tag{6}
$$

In Table 1, the column for the T_{i+1} of Y_i shows that an n-digit operand Y_{BCD} could generate an $(n+1)$-digit SD recoded operand Y_{SD}, and the $(n+1)$th digit in Y_{SD} can only be 0 or 1. Thus, for the $(n+1)$th partial product can only be $0X$ (all zeros) or $1X$. Furthermore, since the $(n+2)$th digit of the multiple $1X$ is always zero, and the $(n+1)$th digit of the $1X$ can only be 0 or 1, only 1 bit is enough to represent the most significant two digits in the $(n+1)$th partial product, PP_n. Thus, the selection logic for PP_n could be simplified. Additionally, the actual bit-widths on the output of the PPG are $4 \times (n+2) \times n + (4 \times n + 1)$ for the partial product PP and n for the inversion signal Y_n. The detailed structure of the PPG is shown in Fig. 3.

5 SD Partial Product Reduction

To illustrate the proposed algorithm, a PPR scheme of a 16×16-digit multiplier is implemented and discussed. First, the layout of the partial product array and the basic structure of the PPR unit are introduced. Subsequently, a PPR algorithm based on multioperand SD addition is discussed. Finally, a hardware implementation of the proposed PPR unit for a 16×16-digit multiplier is addressed. Additionally, the delay model in terms of the delay of a binary full adder is analyzed to guide in designing of the proposed SD-BCD converter.

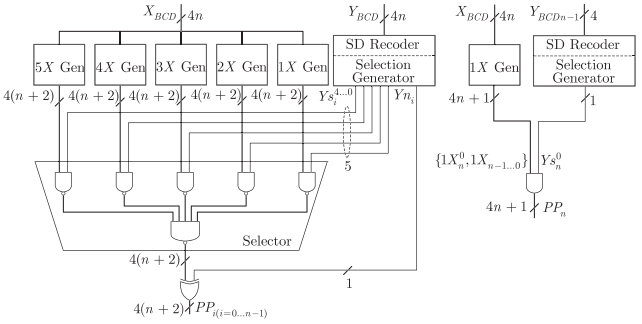


Fig. 3. Proposed architecture of partial product generation.

5.1 Partial Product Array

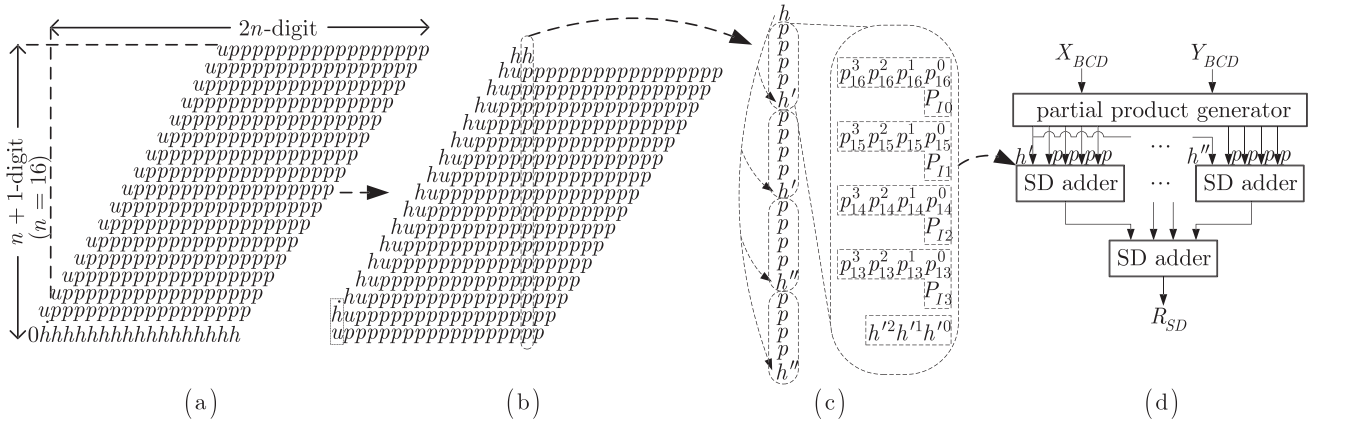
As described in Section 4, for the multiplication of two n-digit operands, $n+1$ partial products in $(n+2)$-digit are generated from the PPG unit in the proposed algorithm. Then, the $n+1$ partial products need to be shifted according to the weight of each digit in the second operand. Finally, these $n+1$ shifted partial products are added by the SD multioperand adders.

An example of the layout of partial products for the proposed 16×16-digit multiplication is shown in Fig. 4a. The partial product $0\dot{h}h\ldots hh$ indicates the partial product generated by the most significant digit (MSD) of the recoded operand Y_{SD}. Recalling the description in Section 4.2, the MSD of Y_{SD} only can be 1 or 0, and the 18th digit of PP_{16} is always zero for a 16×16-digit multiplication. Furthermore, the 17th digit of PP_{16}, $\dot{h}$, is in $[0, 1]$ as shown in Table 1. The $up\ldots pp$ represents the partial products generated according to the least 16 significant digits of Y_{SD}. Since the 18th digit may be ±1 only in $\pm5X$, the range of u is restricted to $[-1, 1]$.

In Fig. 4b, we rearrange the layout of the partial product array. Thus, except the middle two partial product columns, all other columns are not more than 16 digits. For a 16×16-digit decimal multiplication, the result is maximally in 32 digits which is the product of two operands with 16 consecutive nines. If the product in SD format is not going to be used by other SD arithmetic units before converting back to BCD format, then the digits beyond the least 32 digits can be discarded (e.g., the digits in a dashed rectangular). For example, in a 16×16-digit multiplication, the least 33 digits of the SD product can only be in one of the formats shown in (7). Otherwise, after converting back to BCD format, it will be larger than 32 digits

$$
R_{SD} = \begin{cases} 1\overline{d}\ldots, & \text{or} \\ 10\ldots0\overline{d}\ldots, & \text{or} \\ 0d\ldots, & \text{or} \\ 00\ldots0d\ldots. \end{cases} \tag{7}
$$

where the d is a positive decimal digit, and $\overline{d}$ is the negation of d. The range of d is dependent on the digit set applied (e.g., $d \in [1, 5]$ for the digit set $[-5, 5]$).



p: one digit of partial products (PPs), $p \in [-8, 8]$.
h: one digit of the 17^{th} partial product, $h \in [-4, 6]$.
h', h'': combinations of one digit $h, h' \in [-2, 2], h'' \in [0, 1]$.
P_{Ii}: an increment bit of digits in the i^{th} partial product.
$\dot{h}$: the 17^{th} digit of the 17^{th} partital product, $\dot{h} \in [0, 1]$.
u: the most significant digit of a partital product, $u \in [-1, 1]$.

Fig. 4. Restructure of the proposed partial product reduction.

The leading one in the first case in (7) will be reduced by one to form a ten in the less significant digit to cancel out the negative digit $\overline{d}$. The 32nd digit of the SD result should be converted to $(10 + \overline{d})$ or $(10 + \overline{d} - 1)$ only depending on the value of the digits on the same position and less significant positions. In the second case, the 0-sequence on the right side of the leading one should be converted to a 9-sequence to cancel out the first negative digit on its right side with the same manner as mentioned in the first case. In the latter two cases, the most significant positive digit will be converted to d or $d - 1$ depending on less significant digits. Furthermore, the most significant positive digit guarantees that no extra borrow is propagated to its left side. Hence, the conversion of an SD digit only depends on the sign of itself and the less significant digits. Consequently, in all of the cases, the 33rd digit is always zero in the result in BCD format. The details of the conversion algorithm to correctly generate the result in BCD format is discussed in Section 6.

Sixteen partial products can be divided into four groups in which a 4-operand SD adder is applied. Subsequently, four results of the first level of 4-operand SD adders are summed up by a 4-operand SD adder in the second level. In the middle two columns in the partial product array, there are 17 partial products which potentially cause a complicated design. In the proposed algorithm, we recode the 17th operand into four subtle numbers (i.e., h', h'') as shown in Fig. 4c, and issue them into four SD adders in the first level. The maximum number of operands for the first level of SD adder is shown in the dashed circle. Note that the increment signals P_{Ii} for all digits in one partial product are identical. The dataflow of the PPR algorithm is given in Fig. 4d. As shown in Fig. 4d, two levels of SD adders are applied in the proposed PPR unit. Furthermore, as shown in the next section, the multioperand adder to process four p and one h' has the same complexity as the adder for four p.

5.2 Multioperand SD Addition Algorithm

In the proposed multiplier, the $n + 1$ partial products are encoded in SD digit-set $[-8, 8]$ within 4 bits 2's complement number and 1 bit increment. The partial product reduction is indeed a multioperand SD addition. Although in principle, the result of the multioperand SD addition could be in the same digit-set as the input operands, to reduce the number of internal wires, we keep the result of the SD addition in $[-8, 7]$ in which the 1-bit increment signal is removed. An SD addition could be simply summarized into three steps, which are adding operands to get position sum ps_i, extracting transfer digit t_{i+1} and obtaining interim sum $w_i = ps_i - 10t_{i+1}$ (suppose radix is 10), and computing final sum $s_i = w_i + t_i$. Actually, a two-operand SD adder can be applied as the minimum element in the PPR unit, and the position sum is corrected (i.e., $ps_i - 10t_{i+1}$) for each addition. However, the correcting operation is not immediately needed, and can be postponed to reduce the delay and area of the PPR. In Table 2, the cases for multiple operands in the SD addition are shown. The range of ps_i limits the selection of t_i, and the range of w_i cannot be decreased infinitely to cover all the digits in a decimal range [0, 9]. Table 2 shows that as the range of ps_i increases, the ranges of t_i and s_i increase. To restrict the range of s_i in $[-8, 7]$, the maximum number of operands in $[-8, 8]$ is four.

If the t_i and w_i are in the ranges of $[-3, 3]$ and $[-5, 4]$ in the proposed algorithm, the maximum range of the position sum ps_i can reach to $[-35, 34]$. The extra range out of $[-32, 32]$ (i.e., sum of four numbers in $[-8, 8]$) implies that the number of operands of the addition on $[-8, 8]$ may be between 4 and 5. In Fig. 4c, we show that the 17th operand, $h \in [-4, 6]$, is recoded into four parts, and the maximum range of the subtle numbers (i.e., the h' and h'') is $[-2, 2]$. Thus, it is possible to add four operands with the subtle number together without overflow on the number system.

TABLE 2
Analysis of the Number of Operands of SD Addition

#Op.	Range of ps_i	Range of t_i	Range of w_i	Range of s_i
2	$[-16, 16]$	$[-1, 1]$	[-6,6]	[-7,7]
3	$[-24, 24]$	$[-2, 2]$	[-5,5]	[-7,7]
4	$[-32, 32]$	$[-3, 3]$	[-5,4]	[-8,7]
5	$[-40, 40]$	$[-4, 4]$	[-5,4]	[-9,8]

TABLE 3
Proposed SD Addition Algorithm

Addition Steps	SD addition operands		Symbols
	Digit i+1	Digit i	
level1-step1: sum the partial products $ps = p + p + p + p + h'$	$4\times[-8,8]$	$4\times[-8,8]$	p
	$+\ [-2,2]$	$+\ [-2,2]$	h'
	$[-34,34]$	$[-34,34]$	ps
level1-step2: generate t_{i+1} and w_i calculate $s_i = t_i + w_i$	$[-5,4]$	$[-5,4]$	w
	$+\ [-3,3]$	$+\ [-3,3]$	t
	$[-8,7]$	$[-8,7]$	s
level2-step1: sum the four SD results $ps' = s + s + s + s$	$4\times[-8,7]$	$4\times[-8,7]$	
	$[-32,28]$	$[-32,28]$	ps'
level2-step2: generate t'_{i+1} and w'_i, calculate $s'_i = t'_i + w'_i$	$[-5,4]$	$[-5,4]$	w'
	$+\ [-3,3]$	$+\ [-3,3]$	t'
	$[-8,7]$	$[-8,7]$	s'

TABLE 4
Proposed Transfer Digit and Interim Sum Recoder

Recoder in 1st-level SD adder			Recoder in 2nd-level SD adder		
ps_i	t_{i+1}	w_i	ps'_i	t'_{i+1}	w'_i
"1100010"	"011"	"0100"	"1111100"	"011"	"1110"
"1100001"	"011"	"0011"	"1111011"	"011"	"1101"
"1100000"	"011"	"0010"	"1111010"	"011"	"1100"
"1100111"	"011"	"0001"	"1111001"	"011"	"1011"
"1100110"	"011"	"0000"	"1111000"	"010"	"0100"
"1100101"	"011"	"1111"	"1110111"	"010"	"0011"
"1100100"	"011"	"1110"	"1110110"	"010"	"0010"
"0011011"	"011"	"1101"	"1110101"	"010"	"0001"
"0011010"	"011"	"1100"	"1110100"	"010"	"0000"
"0011001"	"011"	"1011"	"1110011"	"010"	"1111"
"0011000"	"010"	"0100"	"1110010"	"010"	"1110"
.....					
"0100110"	"101"	"1100"	"0100000"	"101"	"1110"

The process of the SD addition according to our proposed number system is listed in Table 3. In the proposed SD addition, the operands are summed up with binary arithmetic. To do the decimal correction, a recoder which maps the binary position sum ps (ps') to the decimal transfer digit t (t') and interim sum w (w') is applied in each level of SD addition.

Since the signed digit operands are involved in the multioperand addition, the addition algorithm of weighted bit-set encoding proposed in [26] is applied and extended for multiple operands and multiple bit-widths in our algorithm. In Fig. 5, the proposed two levels of SD additions are illustrated by the dot notation representation which is proposed in [26]. In Fig. 5, the white circle represents a binary bit with negative weight, namely negabit, and the black circle represents a binary bit with positive weight, namely posibit. Additionally, the carry save half adder (HA), full adder, and 4:2 compressor are, respectively, represented by the dashed rectangles with two, three, and four circles. The solid line, solid double-line, and bold solid line represent one level of carry save arithmetic units, a carry lookahead adder (CLA), and a recoder, respectively.

As shown in Fig. 5, the transfer digits and interim sums from the first level of SD addition are summed up directly

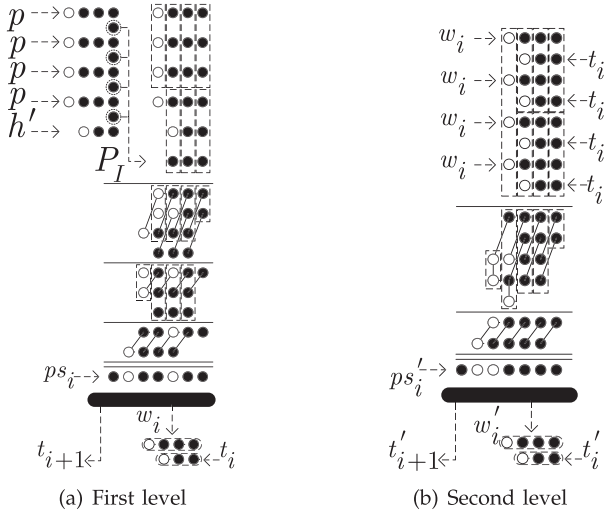


Fig. 5. Dot notation of the proposed two levels of multioperand SD additions.

in the second level of SD addition to avoid the delay cost of a carry lookahead adder to add w and t. Therefore, the step 2 of the first level of addition and the step 1 of the second level of addition proposed in Table 3 are merged together. Furthermore, to reduce the number of the arithmetic units in the hardware implementation, we do not extend the sign bit of the operands (i.e., h' and P_I). Thus, the position sum ps (ps') is given in hybrid posibit-negabit encoding. For example, the third bit and sixth bit of ps have negative weight -2^2 and -2^5. Note that in Fig. 5a, the increment signal P_I for each digit is summed up by a binary counter to reduce the number of operands in the least significant bit of each SD adder. Such a counter can be applied right after the Radix-10 operand recoder of the operand Y, thus it cannot affect the critical path. Additionally, since the increment bits for all digits in a partial product are identical, the number of the counters can be minimized.

The hybrid posibit-negabit encoded binary to signed digit decimal recoder which is a one-to-one mapping can be implemented in the combinational logic. A segment of the map in binary bits to recode ps_i and ps'_i is given in Table 4. As shown in Fig. 5, the ps is represented in hybrid posibit-negabit encoding, and the negative weighted bits are placed at the third and sixth binary position. Thus, in the recoder, an input of "1100010" (34) generates "011" (3) as t and "0100" (4) as w.

5.3 Hardware Implementation and Delay Model of the Proposed PPR

As shown in Fig. 5a, the maximum bits of operands of the first level SD adder are six, which can be reduced to one carry-sum pair by three levels of binary full adders (FA) and half adders as shown in Fig. 6. By applying the WBS adder, the inverters are placed on the input or output of the traditional arithmetic unit, such as a full adder. As shown in [27], [28], and [26], the inverters in between the arithmetic units can be canceled. The remaining inverters at the input and output of the calculation unit could be absorbed by the previous logic. For example, the inverters of the negabits $\overline{p}^3_{i...i+3}$ can be removed by the XOR gates at the output port of the PPG with inverted logic (i.e., XNOR gate). To save the delay on the critical path, we keep the transfer digit t_i and the interim sum w_i generated by the first level of SD adders, and sum up all those eight internal parameters by two

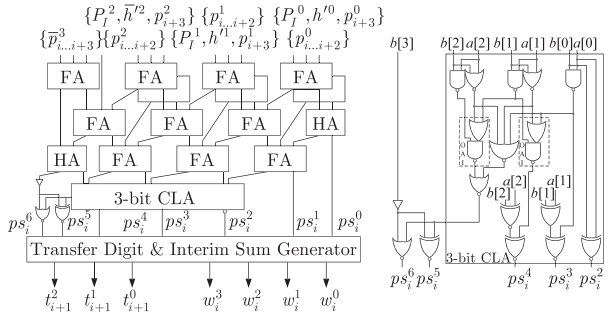


Fig. 6. Hardware structure of the proposed first-level multioperand SD adder.

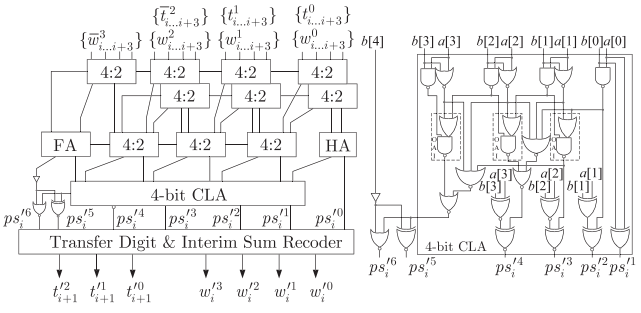


Fig. 7. Hardware structure of the proposed second-level multioperand SD adder.

levels of binary 4:2 compressors as shown in Fig. 7. Since the recoder inside the PPR unit is a simple one-to-one mapping from the inputs to the outputs, the recoders described in Table 4 are simply created by the combinational logic gates. Note that, except two middle columns of operands as shown in Fig. 4b, all other columns can be reduced with elements no more complicated than the adders on the critical path which are shown in Figs. 6 and 7. For example, 12 operands can be reduced by four 3-operand SD adders on the first level and one 4-operand SD adder on the second level. Thus, the area of the PPR is potentially reduced. Finally, a segment of the top-level architecture of the SD adders in the PPR unit for a 16×16-digit multiplier is given in Fig. 8.

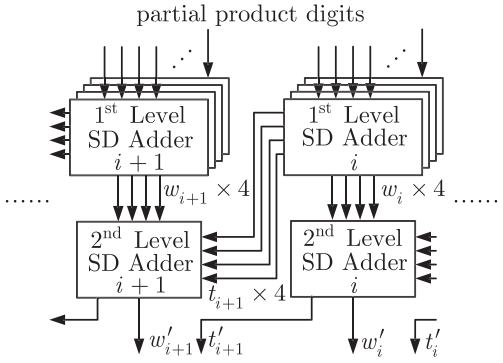


Fig. 8. Top-level architecture of the proposed partial product reduction unit.

In addition, the different structures of columns of PPR unit make the result signals of different digits of the PPR available at different time. To analyze the delay on each digit of the output of the PPR, a list of equivalent binary full adders in modules on the critical paths in each column is shown in Table 5. We assume that the binary 4:2 compressor has a delay of 1.5 binary full adder, and the binary 5:2 compressor has a delay which equals to 2 BFAs' delay [24]. According to the delay analysis, we assume that the 3-bit and 4-bit carry lookahead adder have delay of 1 BFA and 1.25 BFAs on the critical path which passes through ps_i^4 and $ps_i'^4$, respectively. The delay of the combinational recoders is also represented in terms of the 3:2 BFA which is obtained by the delay analysis. Thus, we could obtain the brief estimation of the delay on each digit of PPR in terms of the equivalent binary full adders. In Table 5, the delay from connected neighbor columns is considered. Additionally, since the latency to generate each partial product in PPG and the delay of the CLA to add w' and t' for each digit are almost the same, we do not consider the influence of the delay of the PPG stage and the final CLA in Table 5.

For a 34×34-digit multiplication, at most 35 partial products should be reduced. The 35 partial produces could be divided into three groups (i.e., double 17 partial products

TABLE 5
Delay Analysis of Each Digit of the Proposed Partial Product Reduction

Logic Module	Column Position															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Bin 3:2	1	1		3	3	3	3								3	3
Bin 4:2*	1.5	1.5	3	1.5	1.5	1.5	1.5	3	4.5	4.5	3	3	3	3	3	3
Bin 5:2*								2			2	2	2	2		
3-bit CLA*	2	2	1	1	1	1	1								1	1
4-bit CLA*			1.25	1.25	1.25	1.25	1.25	2.5	2.5	2.5	2.5	2.5	2.5	2.5	1.25	1.25
Recoders*	2.5	2.5	2.75	3	3	3	3	3	3	3	3	3	3.25	3.25	3.5	3.5
equivalent BFAs	7	7	8	9.75	9.75	9.75	9.75	10.5	10	10	10.5	10.5	10.75	10.75	11.75	11.75

Logic Module	Column Position															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bin 3:2								3							1	
Bin 4:2*	3	3	3	3	4.5	4.5	4.5		1.5	1.5	3	1.5		1.5		
Bin 5:2*	2	2	2	2				2	2	2			2			
3-bit CLA*								1	1	1	1				1	
4-bit CLA*	2.5	2.5	2.5	2.5	2.5	2.5	2.5	1.25	1.25	1.25	1.25	2.5	1.25	1.25		
Recoders*	3.25	3.25	3	3	3	3	3	3	2.5	2.5	2.75	2.5	1.5	1.5	1.25	1
equivalent BFAs	10.75	10.75	10.5	10.5	10	10	10	10.25	8.25	8.25	8	6.5	4.75	4.25	3.25	1

* The delay is represented in the number of equivalent BFAs.

and one extra partial product). For the double 17 partial products, we have shown that the proposed structure could be applied to obtain two SD results. Thus, one more level of 3-operand SD adders are applied on the critical path to reduce the two SD results of the $17:2$ SD addition and the extra one partial product.

6 SD-BCD Conversion

The partial products in a signed digit-set can be reduced to one SD result with the multioperand SD adders. Unlike in other works, we propose a digit-set converter to convert back the SD result into the conventional BCD encoding. Moreover, in such an SD-BCD conversion algorithm, a hybrid carry propagation network is discussed in detail.

6.1 SD-BCD Conversion Algorithm

In the proposed multiplier, the $2n$-digit result of the PPR is in digit-set $[-8,7]$. If the digit is negative, a borrow (i.e., negative carry) occurs. To convert it back to the digit set $[0,9]$ in BCD encoding, the negative digit is increased by 10, and the first nonzero digit with higher weight is reduced by one. All the continuous zeros in between the current negative digit and the first nonzero digit on its left side are converted to 9. An example is provided below:

$$(100\overline{4}802\overline{3})_{SD} = (09952017)_{BCD}.$$

Thus, to convert the SD result into BCD encoding, the negative digit (i.e., generates the negative carry) and zero digit (i.e., propagates the negative carry) need to be detected. Furthermore, a carry propagation network and necessary logics are applied to determine and convert the SD digits into BCD encoding. The conversion algorithm is shown as follows:

Algorithm I. SD to BCD Encoding Conversion
Input: SD number S.
Output: BCD number R.
1. Detect borrow generation bit (G_i) and propagation bit (P_i) for each digit of S.

$$G_i = \begin{cases} 1 & \text{if } S_i < 0 \\ 0 & \text{otherwise,} \end{cases} \quad P_i = \begin{cases} 1 & \text{if } S_i = 0 \\ 0 & \text{otherwise,} \end{cases}$$
$$G_{i:j} = \begin{cases} G_i & \text{if } i = j \\ G_i + (P_i \cdot G_{i-1:j}) & \text{if } i > j, \end{cases} \quad (8)$$
$$P_{i:j} = \begin{cases} P_i & \text{if } i = j \\ P_i \cdot P_{i-1:j} & \text{if } i > j. \end{cases}$$

2. Compute the negative carry C_i of S ($C_0 = 0$).

$$C_{i+1} = G_{i:j} + (P_{i:j} \cdot C_j) \quad (9)$$

3. Convert the result S to BCD encoding.

$$R_i = \begin{cases} S_i & \text{if } C_{i+1}C_i = 00 \\ S_i - 1 & \text{if } C_{i+1}C_i = 01 \\ S_i + 10 & \text{if } C_{i+1}C_i = 10 \\ S_i + 9 & \text{if } C_{i+1}C_i = 11 \end{cases} \quad (10)$$

6.2 Hardware Implementation of the Converter

In Algorithm I, the first step is to detect the negative and zero digits. Since in the proposed multiplier, the outputs of the PPR can be added into an SD number in the 4-bit two's complement encoding, the negative detection is simply a fourth-bit detection. To detect a zero digit in two's complement encoding, all four bits are needed. Since inside the 4-bit CLA to sum up the final transfer digit t' and interim sum w', the results on different bits in a digit are available at different time, only one extra OR gate on critical path for the zero detection could be achieved by connecting three OR gates in cascade as shown in Fig. 10.

For the traditional method in the carry propagation step, a $\lceil \log(an) \rceil$-level prefix network is applied to quickly generate the final carry. The parameter a depends on the processing scope (e.g., in [22] the proposed quaternary tree unit works in bit level, thus a $\lceil \log(4n) \rceil$-level prefix tree is applied). No matter how many levels are in the prefix tree, the critical path passes through all levels of internal nodes. On the other hand, in the PPR stage, the longest path is potentially on the middle columns of the partial product array, and the rest of columns have shorter paths. It implies that the digits in final product which are close to the least and most significant digits are available earlier and can be processed before the digits in the middle part of the partial products array are ready. In Section 5.3, a delay model on each digits of the final product is shown in Table 5. According to the estimated delay, we divide the 32-digit SD result into five groups which are $gp_0 = \{S_{11}, \ldots, S_0\}$, $gp_1 = \{S_{15}, \ldots, S_{12}\}$, $gp_2 = \{S_{17}, S_{16}\}$, $gp_3 = \{S_{21}, \ldots, S_{18}\}$, and $gp_4 = \{S_{31}, \ldots, S_{22}\}$. For each group, a small traditional carry propagation tree is applied. Thus, the well-optimized prefix tree circuit for binary design could be reused. The carry propagation process is described in the following equations:

$$C_i = \begin{cases} 0 & \text{if } i = 0 \\ G_{i-1:0} & \text{if } 12 \geq i \geq 1 \\ G_{i-1:12} + P_{i-1:12} \cdot C_{12} & \text{if } 16 \geq i \geq 13 \\ G_{i-1:16} + P_{i-1:16} \cdot C_{16} & \text{if } 18 \geq i \geq 17 \\ G_{i-1:18} + P_{i-1:18} \cdot C_{18} & \text{if } 22 \geq i \geq 19 \\ G_{i-1:22} + P_{i-1:22} \cdot C_{22} & \text{if } 32 \geq i \geq 23, \end{cases} \quad (11)$$

where the C_i is the carry-in of the i digit, and note that the carry-in to the least significant digit is always zero.

In Fig. 9, a detailed structure of the proposed prefix network is shown. The white dot represents the logic to create the generation bit G_i and propagation bit P_i for each digit. The black dot represents the logic to create the group generation bits $G_{i:j}$ and group propagation bits $P_{i:j}$ described in (8). For the low 12 digits, a Ladner-Fischer network is applied to minimize the number of levels and the area cost. Since the carry-in on the least significant digit is always zero, the carry-in to 13th digit equals to $G_{11:0}$. For the digits from S_{12} to S_{15}, a two-level Ladner-Fischer network is used to create the group-carry-in generation and propagation signal, $G_{15:12}$ and $P_{15:12}$. To further calculate the carry, only an AND-OR gate is needed. For carry C_{18} and C_{17}, a 2-bit carry look-ahead structure is used. In high 14 digits, we use the same technique as the one in the low 16 digits. Note that to reduce the fan-out of gates from low weight inputs through high weight outputs, a Han-Carlson network is applied to calculate the group-carry propagation and generation signals. In the 16-digit multiplication, at least 5-level of internal nodes should be on the critical path in a conventional method. In the proposed architecture,

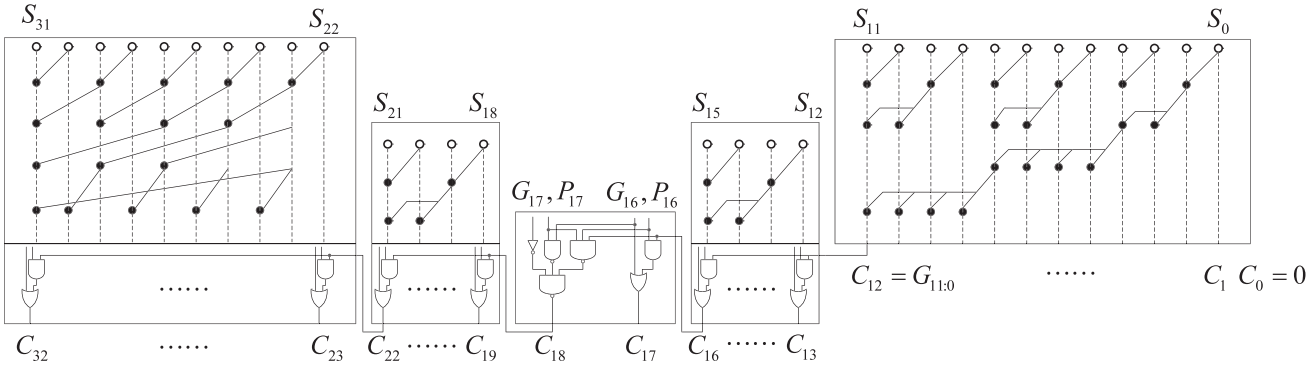


Fig. 9. Proposed hybrid prefix network in the SD-BCD converter.

about 3-level of nodes are connected after the outputs of the middle columns in partial products array, and the level of nodes after the most significant columns are kept as 5. Although for less significant columns, the connected prefix tree nodes would be greater than five, the shorter delay on those columns could counterbalance the delay of the nodes in the prefix network. Note that the architecture of the hybrid prefix network highly depends on the structure of PPR. An improved structure would provide a better performance if the PPR structure is changed.

In the third step of the Algorithm I, the SD result which is converted into BCD encoding by the conditional adder is selected by the carry signals of two neighbor digits in S. To convert the SD result into BCD encoding, since the correction signals (i.e., "0000," "1111," "1010," and "1001,") for the four different carry-in cases are constant, the correction process could be designed as a conditional constant addition which could be comparatively simplified. In Fig. 11, the circuit of one digit conditional constant adder which consists of three constant adder and a combinational selector is shown.

7 Performance Evaluation and Comparison

To compare the proposed multiplication algorithm with other designs, we first create a delay model in terms of number of 2-input NAND gates on the estimated critical path. Thus, the effects from fan-out gates and the gate scaling are ignored in the theoretical comparison. To obtain a more accurate comparison, a Verilog-HDL model of the proposed 16×16-digit multiplier is synthesized with STM 90 nm standard cell library. A discussion on the differences of performance between our proposed architecture and other designs is given afterwards.

7.1 Performance Evaluation

In Table 6, the numbers of logic gates (i.e., NAND2 gate or ΔG) for different stages of the parallel 16×16-digit multipliers from other designs are listed. We assume that an AND2/OR2 gate equals to one NAND2 gate, and an XOR gate equals to two NAND2 gates. The PPG unit in Table 6 is used to generate the partial products in the format which can be directly processed by the PPR unit in the next stage. For example, the decimal carry save adder, to reduce the multiples from double-BCD format (i.e., double-four-bit) to BCD-CS format (i.e., one-four-bit) applied in the sequential design in [21] and the parallel design in [12], is counted into the PPG stage. Additionally, to fairly analyze the efficiency of the PPR methods, we suppose that the outputs of the PPR unit are two numbers in arbitrary formats (e.g., double-BCD or BCD-CS format). Thus, the fourth level of the ODDS adder in [25] and the final simplified CLA shown in Fig. 10 are assumed as the adder setup unit in the final stage. Finally, for the three sequential multipliers in the bottom of Table 6, we only provide the ratio (e.g., marked by an asterisk) between the ΔG involved in iterative cycles and the proposed design, since other noniterative cycles can be pipelined without reducing the overall efficiency of the multiplier. As shown in Table 6, some algorithms may be faster than our proposed design on PPG or PPR, but by considering the tradeoff among three multiplication stages, our design can benefit on the overall performance.

TABLE 6
Delay Analysis of 16×16-Digit Decimal Fixed-Point Multipliers

Architecture		PPG (ΔG)	Ratio	PPR (ΔG)	Ratio	Setup+Final Adder (ΔG)	Ratio	Total (ΔG)	Ratio
Parallel	[12]	20	1.67	60	1.28	17	1.06	97	1.29
	[21]	37	3.08	54	1.15	25	1.56	116	1.55
	[15]	9	0.75	57	1.21	19	1.19	85	1.13
	[25]	11	0.92	46	0.98	29	1.81	86	1.15
	Radix-10 [16]	39	3.25	42	0.89	23	1.44	104	1.39
	Radix-5 [16]	11	0.92	53	1.13	23	1.44	87	1.16
	Proposed	12	1	47	1	16	1	75	1
Sequential	[11]	11	-	13×17	-	43	-	275	2.95*
	[10]	13	-	31×17	-	-	-	-	7.03*
	[21]	20	-	20×17	-	17	-	377	4.76*

* Ratio=$\Delta_{PPR}/\Delta_{proposed\ total}$

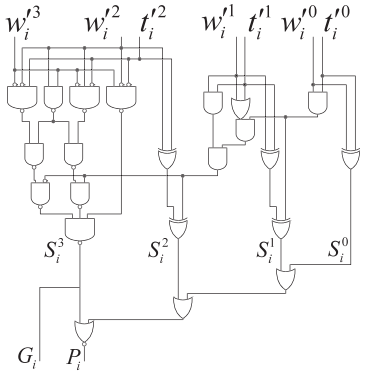


Fig. 10. Simplified 4-bit CLA and G, P generation circuit.

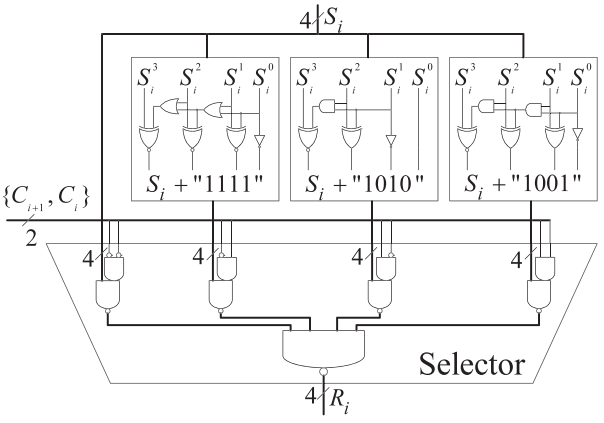


Fig. 11. Final conditional constant adder.

To obtain a more accurate performance on not only timing delay but also hardware cost, a hardware model for a 16×16-digit multiplier is implemented by Verilog-HDL and synthesized using Synopsys Design Compiler and STM 90 nm CMOS standard cells library. Five hundred thousand random cases and 100 manually created boundary cases are verified in the Verilog-HDL model to guarantee the correctness. The FO4 delay of this library is 45 ps under the typical condition (1V, $25°$C). The delay in picosecond of each module on the critical path is shown in Table 8. Furthermore, the delay-area values which are measured under Design Compiler within the range from 1.94 ns and 49,900 NAND2 to 2.65 ns and 36,655 NAND2 are shown in Fig. 12. The delay-area values of other parallel designs are also provided. The latest designs of the architectures Radix-10 and Radix-5 in [16] and the architecture in [25] are implemented and evaluated with our library and synthesis environment.

TABLE 8
Critical Path of the Proposed 16×16-Digit Multiplier

Gen. of mult.	Sel.+Inv.	PPR	GP gen.+Prefix Tree+Sel.
$160ps$	$140ps$	$1230ps$	$410ps$

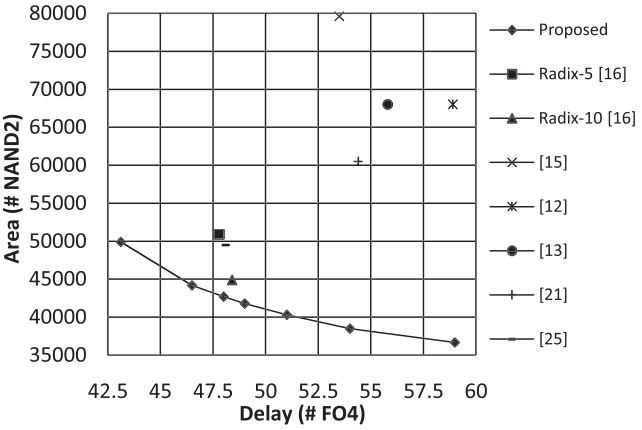


Fig. 12. Delay-area space of the decimal multipliers.

7.2 Comparison and Discussion

In Table 7, we list the state-of-the-art decimal multipliers for 16-digit operands in terms of timing delay, hardware area, throughput, and latency. In [12], the design is synthesized using the STM 90 nm library which is the same library as we used. The latency provided by the authors is 2.65 ns, which equals to about 58.9 FO4. In [13], the authors improve the design in [12] and reduce the latency to 2.51 ns (55.8 FO4) by an elaborated PPR tree and a binary to decimal converter. Both of these designs have the area of 68,000 NAND2 gate. Our PPG algorithm avoids the decimal CSA in the PPG unit applied in those designs. Furthermore, the PPG which consists of six levels of BCD-FAs in [12] involves six levels of carry propagations in 4-bit width which lower the performance of the multiplier. These two radix-10 combinational multipliers costs at least 29 percent more timing delay and 36 percent bigger area than our proposed design.

TABLE 7
Performance Comparison of 16×16-Digit Decimal Fixed-Point Multipliers

Architecture		# Cycles	Cycle Time	Latency		Throughput	Area	
			(FO4)	(FO4)	Ratio	Mult./Cycle	(NAND2)	Ratio
Parallel	Bin. Radix-4 [12]	1	-	31.1	0.72	1	34000	0.68
	[12]	1	-	58.9	1.37	1	68000	1.36
	[13]	1	-	55.8	1.29	1	68000	1.36
	[21]	1	-	54.4	1.26	1	60500	1.21
	[15]	1	-	53.5	1.24	1	79600	1.60
	[25]	1	-	48.1	1.12	1	49500	0.99
	Radix-10 [16]	1	-	48.4	1.12	1	44400	0.89
	Radix-5 [16]	1	-	47.8	1.11	1	50900	1.02
	Proposed	1	-	43.1	1	1	49900	1
Sequential	[11]	24	12.7	305	5.00*	1/17	31500	0.63
	[10]	20	16	320	6.31*	1/17	16000	0.32
	[21]	20	14.7	294	5.80*	1/17	18550	0.37

* Ratio=$(FO4_{Delay}/Throughput)/FO4_{proposed}$

In [21], the authors propose a parallel decimal floating-point multiplier by applying the fixed-point design with radix-10 architecture proposed in [23]. Such a parallel decimal multiplier applies new decimal encodings (i.e., BCD-4221 and BCD-5211) to simplify the design of the PPR tree. In the proposed radix-10 design in [23], a carry propagation through all bits in an operand is involved in the PPG stage. Besides, the proposed 17:2 reduction tree with binary CSAs and encoding converters is slower than our proposed PPR unit with two levels of SD adders. Overall, our proposed algorithm reduces about 26 percent timing delay and 21 percent area compared to the radix-10 design applied in [21].

In [15], the authors propose a method to represent $8X$ and $9X$ in two digits to avoid the long path in PPG. Consequently, the delay of the PPG is reduced significantly. To reduce the partial products, the authors present an architecture within 6-level simplified BCD-FA. Additionally, after the PPR unit, a narrower result is obtained. However, the level of prefix tree applied in the final addition cannot be reduced, since the reduction on the result of PPR is not over half of the width. The BCD-FA used in the PPG in [12] is replaced by a simplified BCD half adder. Nevertheless, the digit-level reduction tree-based BCD-FA shows the disadvantage of the relatively large delay and big area as we described for the design proposed in [12]. The synthesized design in [15] under TSMC 130 nm standard cells library costs about 53.5 FO4 and 79,600 NAND2, respectively. Although the PPG unit which has no XOR gate and a simple selection circuit is faster than our proposed PPG, due to the slower PPR and final addition in [15], our multiplier could gain about 24 percent less delay with 60 percent less hardware cost overall.

In the SD multiplier proposed in [25], the efficiency of the PPG and PPR units are guaranteed as in our proposed design. However, the double-BCD format partial product array takes off the advantage of the overall performance on timing delay in [25]. Furthermore, the final overloaded decimal digit set adder with the following traditional digit converter are slower than the simplified 4-bit CLA and the converter proposed in our design. Finally, the synthesis result shows that our design takes 12 percent less delay with 1 percent more area cost.

In [16], the authors improve the design they proposed in [23]. The PPR trees are optimized for both radix-10 and radix-5 architectures in [23]. Thus, as shown in Table 6, the number of gates for the 17:2 reduction tree in radix-10 architecture is faster than our proposed PPR. However, the 32:2 reduction tree for radix-5 architecture is still slower than our design, since about twice operands are processed in such a structure. Moreover, the carry propagation which affects the overall performance in the PPG of the radix-10 architecture cannot be avoided. In the radix-5 architecture, the partial products are generated by a couple of recoders within a small delay. Additionally, the unconventional encodings avoid the complicated decimal correction in most of other works. Thus, the proposed PPR tree could be arranged as the binary CSA tree (i.e., Wallace-like structure based on binary CSAs and encoding converters). However, our design balances the delay of PPG and PPR and applies a simpler final conversion compared to the designs in [16]. Overall, our design has about 11 percent less delay with 2 percent less area compared to the fastest state-of-the-art design (radix-5) in [16], and has about 12 percent less delay with 11 percent more area compared to the radix-10 architecture in [16].

The sequential designs of the fixed-point decimal multiplication are also listed in Table 7. The latency ratio with asterisk is calculated according to the FO4 spent on iterative cycles. Such sequential designs show the advantage on the area cost and disadvantage on latency and throughput as expected.

Generally speaking, the output format of a PPG algorithm can be a single-BCD (e.g., the radix-10 architecture in [16]), a single-BCD with identical carry for each partial product (e.g., the proposed method), a BCD-CS (e.g., the method applied in [12]), or a double-BCD (e.g., the algorithms used in [15], [25], and the radix-5 architecture in [16]). In general, the less bits in the output of a PPG, the more complexities in the PPG, and the less complexities in the PPR. For example, the single-BCD result of the PPG in the radix-10 architecture in [16] provides the chance to apply the simplest PPR unit, but it cannot avoid the carry propagation in the PPG. On the other hand, the double-BCD result of the simplest PPG in [15] involves a complicated PPR unit. Our proposed PPG method generates the partial product which has the bit-width close to the single-BCD format without the carry propagation. Thus, the complexity of the PPR is potentially reduced. Moreover, since only simple combinational logic is applied to convert the digit-set in the proposed PPG, the BCD-FA used in the PPG of [12] is eliminated. The PPR algorithm highly depends on the encoding of the result of the PPG. Besides, in a PPR, the less input width the better, and the less bit-level carry propagation the better. Our proposed PPR design based on the multioperand SD addition which involves two bit-level carry propagation is a bit more complicated than the design in [16] with the same input width (i.e., $n+1$ digits), and is simpler than the design in [16] with the double-sized input width and the designs based on BCD-FA in [15] and [12]. The carry propagation in the final addition cannot be avoided in any method, since the result of the PPR is in the redundant format. However, the efficiency of the final addition or conversion can be affected by the complexity of the setup logic and the prefix tree. The proposed conversion method involves a 4-bit carry propagation to generate the propagation and generation bit for each digit, but by applying the hybrid carry prefix tree, the logic on the critical path is minimized. After all, although the proposed method is not the simplest on some stages of a multiplication, the overall delay of the proposed multiplication is minimized by considering the tradeoff of the complexity in each stage.

8 Conclusions

In this paper, we have proposed a technique to implement the parallel decimal multiplication. Unlike other designs, in the proposed algorithm, the multiples (i.e., from $-5X$ to $5X$) are represented in a redundant digit-set $[-8, 8]$. Thus, the $n+1$ SD partial products could be generated without the carry propagation in $3X$. To reduce the $n+1$ partial products, an $(n+1):2$ partial product reduction unit based on the multioperand SD addition is discussed. All the components inside the multioperand SD adder except two

combinational recoders could reuse the hardware in binary designs. To further improve the performance, the proposed hybrid prefix network shows the advantage to squeeze more delay from the critical path in the final digit-set conversion. Overall, the synthesis result under STM 90 nm technology shows that the proposed design could achieve about 11 percent less delay with 2 percent less hardware cost compared to the fastest state-of-the-art parallel decimal multiplier.

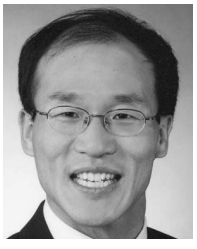
Acknowledgments

The authors would like to acknowledge the anonymous reviewers involved in the review of this manuscript. This project is supported by the Electrical and Computer Engineering Department in University of Saskatchewan and the Natural Science and Engineering Research Council (NSERC) of Canada.

References

[1] M.F. Cowlishaw, "Decimal Floating-Point Algorism for Computers," *Proc. 16th IEEE Symp. Computer Arithmetic,* pp. 104-111, June 2003.
[2] *IEEE Standard for Floating-Point Arithmetic,* IEEE Working Group of the Microprocesser Standards Subcommittee, 2008.
[3] F.Y. Busaba et al., "The IBM z900 Decimal Arithmetic Unit," *Proc. Conf. Record 35th Asilomar Conf. Signals, Systems and Computers,* vol. 2, pp. 1335-1339, 2001.
[4] E.M. Schwarz, J.S. Kapernick, and M.F Cowlishaw, "Decimal Floating-Point Support on the IBM System z10 Processor," *IBM J. Research and Development,* vol. 53, no. 1, pp. 4:1-4:10, Apr. 2010.
[5] L. Eisen et al., "IBM POWER6 Accelerators: VMX and DFU," *IBM J. Research and Development,* vol. 51, no. 6, pp. 1-21, 2007.
[6] M. Cornea et al., "A Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic Using the Binary Encoding Format," *IEEE Trans. Computers,* vol. 58, no. 2, pp. 148-162, Feb. 2009.
[7] L.-K. Wang et al., "A Survey of Hardware Designs for Decimal Arithmetic," *IBM J. Research and Development,* vol. 54, no. 2, pp. 8:1-8:15, 2010.
[8] L-K. Wang et al., "Benchmarks and Performance Analysis of Decimal Floating-Point Applications," *Proc. 25th Int'l Conf. Computer Design,* pp. 164-170, Oct. 2007.
[9] M.A. Erle and M.J. Schulte, "Decimal Multiplication via Carry-Save Addition," *Proc. IEEE Int'l Conf. Application Specific Systems, Architectures, and Processors,* pp. 348-358, June 2003.
[10] M.A. Erle, E.M. Schwarz, and M.J. Schulte, "Decimal Multiplication with Efficient Partial Product Generation," *Proc. 17th IEEE Symp. Computer Arithmetic,* pp. 21-28, 2005.
[11] R.D. Kenney, M.J. Schulte, and M.A. Erle, "A High-Frequency Decimal Multiplier," *Proc. IEEE Int'l Conf. Computer Design: VLSI in Computers and Processors,* pp. 26-29, 2004.
[12] T. Lang and A. Nannarelli, "A Radix-10 Combinational Multiplier," *Proc. 40th Asilomar Conf. Signals, Systems and Computers,* pp. 313-317, Oct. 2006.
[13] L. Dadda and A. Nannarelli, "A Variant of a Radix-10 Combinational Multiplier," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '08),* pp. 3370-3373, May 2008.
[14] L. Dadda, "Multioperand Parallel Decimal Adder: A Mixed Binary and BCD Approach," *IEEE Trans. Computers,* vol. 56, no. 10, pp. 1320-1328, Oct. 2007.
[15] G. Jaberipur and A. Kaivani, "Improving the Speed of Parallel Decimal Multiplication," *IEEE Trans. Computers,* vol. 58, no. 11, pp. 1539-1552, Nov. 2009.
[16] A. Vázquez, E. Antelo, and P. Montuschi, "Improved Design of High-Performance Parallel Decimal Multipliers," *IEEE Trans. Computers,* vol. 59, no. 5, pp. 679-693, May 2010.
[17] I.D. Castellanos and J.E. Stine, "Decimal Partial Product Generation Architectures," *Proc. 51st Midwest Symp. Circuits and Systems,* pp. 962-965, Aug. 2008.
[18] I.D. Castellanos and J.E. Stine, "Compressor Trees for Decimal Partial Product Reduction," *Proc. 18th ACM Great Lakes Symp. VLSI,* pp. 107-110, May 2008.
[19] M.A. Erle, M.J. Schulte, and B.J. Hickmann, "Decimal Floating-Point Multiplication via Carry-Save Addition," *Proc. 18th IEEE Symp. Computer Arithmetic,* pp. 25-27, 2007.
[20] B.J. Hickmann, A. Krioukov, M.J. Schulte, and M.A. Erle, "A Parallel IEEE P754 Decimal Floating-Point Multiplier," *Proc. IEEE Int'l Conf. Computer Design,* pp. 296-303, 2007.
[21] M.A. Erle, B.J. Hickmann, and M.A. Schulte, "Decimal Floating-Point Multiplication," *IEEE Trans. Computers,* vol. 58, no. 7, pp. 902-916, July 2009.
[22] A. Vázquez and E. Antelo, "Conditional Speculative Decimal Addition," *Proc. Seventh Conf. Real Numbers and Computers (RNC 7),* pp. 47-57, July 2006.
[23] A. Vázquez, E. Antelo, and P. Montuschi, "A New Family of High-Performance Parallel Decimal Multipliers," *Proc. 18th IEEE Symp. Computer Arithmetic,* pp. 195-204, June 2007.
[24] C.H. Chang, J. Gu, and M. Zhang, "Ultra Low-Voltage Low-Power CMOS 4-2 and 5-2 Compressors for Fast Arithmetic Circuits," *IEEE Trans. Circuits and Systems I: Regular Papers,* vol. 51, no. 10, pp. 1985-1997, Oct. 2004.
[25] S. Gorgin and G. Jaberipur, "A Fully Redundant Decimal Adder and Its Application in Parallel Decimal Multipliers," *Microelectronics J.,* vol. 40, no. 10, pp. 1471-1481, Oct. 2009.
[26] G. Jaberipur and B. Parhami, "Constant-Time Addition with Hybrid-Redundant Numbers: Theory and Implementations," *Integration, the VLSI J.,* vol. 41, pp. 49-64, 2008.
[27] T. Aoki et al., "Signed-Weight Arithmetic and Its Application to a Field-Programmable Digital Filter Architecture," *IEICE Trans. Electronics,* vol. E82-C, no. 9, pp. 1687-1698, 1999.
[28] P. Kornerup, "Reviewing 4-to-2 Adders for Multi-Operand Addition," *J. VLSI Signal Processing,* vol. 40, pp. 143-152, 2005.
[29] H. Nikmehr, B.J. Phillips, and C.C. Lim, "A Decimal Carry-Free Adder," *Proc. SPIE Conf. Smart Material Nano-, Micro-Smart Systems,* pp. 786-797, Dec. 2004.
[30] R. Raafat et al., "A Decimal Fully Parallel and Pipelined Floating Point Multiplier," *Proc. 42nd Asilomar Conf. Signals, Systems, and Computers,* Oct. 2008.
[31] R. Samy et al., "A Decimal Floating-Point Fused-multiply-Add Unit," *Proc. 53rd MidWest Symp. Circuits and Systems (MWSCS),* Aug. 2010.
[32] Decimal IP, SilMinds, http://www.silminds.com/decimal-products, 2012.
[33] ANSI C decNumber Library v3.68, http://speleotrove.com/decimal/#decNumber, 2012.
[34] GNU C Compiler Library, http://gcc.gnu.org/onlinedocs/gcc/Decimal-Float.html, 2012.

Liu Han received the BSc degree in computer science at the Harbin University of Science and Technology, China, in 2005. He received the MSc degree in system-on-chip at the Lund University, Sweden, in 2009. Currently, he is working toward the PhD degree in the Department of Electrical and Computer Engineering at the University of Saskatchewan, Canada. His research interests include computer arithmetic, computer architecture, and VLSI design. He is a student member of the IEEE.



Seok-Bum Ko received the PhD degree in electrical and computer engineering at the University of Rhode Island, Kingston, Rhode Island, in 2002. He is currently an associate professor in the Department of Electrical and Computer Engineering, University of Saskatchewan, Saskatoon, Canada. He worked as a member of technical staff for Korea Telecom Research and Development Group, Korea, from 1993 to 1998. His research interests include computer arithmetic, digital design automation, and computer architecture. He is a senior member of the IEEE and the IEEE computer society.