Sign-Magnitude Encoding for Efficient VLSI Realization of Decimal Multiplication

Saeid Gorgin, Ghassem Jaberipur

Abstract*—Decimal $X \times Y$ multiplication is a complex operation, where intermediate partial products (IPP) are commonly selected from a set of pre-computed radix-10 X-multiples. Some works require only $[0,5] \times X$ via recoding digits of Y to one-hot representation of signed-digits in $[-5,5]$. This reduces the selection logic at the cost of one extra IPP. Two's complement signed-digit (TCSD) encoding is often used to represent IPPs, where dynamic negation (via one XOR per bit of X-multiples) is required for the recoded digits of Y in $[-5,-1]$. In this paper, despite generation of 17 IPPs, for 16-digit operands, we manage to start the partial product reduction (PPR) with 16 IPPs that enhances the VLSI regularity. Moreover, we save 75% of negating XORs via representing pre-computed multiples by sign-magnitude signed-digit (SMSD) encoding. For the first level PPR, we devise an efficient adder, with two SMSD input numbers, whose sum is represented with TCSD encoding. Thereafter, multi-level TCSD 2:1 reduction leads to two TCSD accumulated partial products, which collectively undergo a special early initiated conversion scheme to get at the final BCD product. As such, a VLSI implementation of 16×16-digit parallel decimal multiplier is synthesized, where evaluations show some performance improvement over previous relevant designs.

***Index Terms*—Radix-10 multiplier, Redundant representation, Sign-magnitude signed digits, VLSI design.**

I. Introduction

DECIMAL arithmetic hardware is highly demanded for fast processing of decimal data in monetary, web based, and human interactive applications [1]. Fast radix-10 multiplication, in particular, can be achieved via parallel partial product generation (PPG) and partial product reduction (PPR), which is however, highly area consuming in VLSI implementations. Therefore, it is desired to lower the silicon cost, while keeping the high speed of parallel realization.

Let $\mathcal{P} = X \times Y$ represent an $n \times n$ decimal multiplication, where multiplicand X, multiplier Y, and product $\mathcal{P}$ are normal radix-10 numbers with digits in $[0,9]$. Such digits are commonly represented via binary coded decimal (BCD) encoding. However, intermediate partial products (IPPs) are represented via a diversity of often redundant decimal digit sets and encodings (e.g., $[0,10]$ carry-save [2], $[0,15]$ overloaded decimal [3], [4], $[-7,7]$ signed digit [5], double 4,2,2,1 [6], and $[-8,8]$ Signed-Digit [7]).

S. Gorgin is with the Department of Electrical Engineering and Information Technology, Iranian Research Organization for Science and Technology (IROST), Tehran 3353-5111, Iran (gorgin@irost.ir).
G. Jaberipur is with the Department of Computer Science and Engineering, Shahid Beheshti University, Tehran 19839-63113, Iran (jaberipur@sbu.ac.ir).. The authors are also affiliated with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.

The choice of alternative IPP representations is influential on the PPG, which is of particular importance in decimal multiplication from two points of view: One is fast and low cost generation of IPPs and the other is its impact on representation of IPPs, which is influential on PPR efficiency. Straight forward PPG via BCD digit by digit multiplication [8], [9] is slow, expensive and leads to n double-BCD IPPs for $n \times n$ multiplication (i.e., $2n$ BCD numbers to be added). However, the work of [10] recodes both the multiplier and multiplicand to sign magnitude signed digit (SMSD) representation and uses a more efficient 3-bit by 3-bit PPG. Nevertheless, following a long standing practice [11], most PPG schemes use pre-computed multiples of multiplicand X (or X-multiples). Pre-computation of the complete set $\{0,1,\ldots 9\} \times X$, as normal BCD numbers, and the subsequent selection is also slow and costly. A common remedial technique is to use a smaller less costly set that can be achieved via fast carry-free manipulation (e.g., $\{0,1,2,4,5\} \times X$) at the cost of doubling the count of BCD numbers to be added in PPR; that is n double-BCD IPPs are generated such as $3X = (2X, X)$, $7X = (5X, 2X)$, or $9X = (5X, 4X)$. We offer a summary of PPG and PPR characteristics of several previous relevant works in the Section II (Table I).

The recoding of multiplier's digits, in some relevant works [4], [6], and [7], leads to a carry bit besides the n recoded digits of multiplier, which will generate an extra partial product. This is particularly problematic for parallel multiplication with $n = 16$ (i.e., number of significand's decimal digits according to IEEE standard size of single precision radix-10 floating-point numbers [12]), where the 17 generated partial products require five PPR levels instead of four (i.e., $\log_2 16$). Furthermore, they dynamically negate positive multiples based on the sign of multiplier's recoded digits. This technique reduces the area and delay of logic that selects the X-multiples at the cost of conditionally negating the selected multiples, which requires at least $4n^2$ XOR gates for $n \times n$ multiplication.

In this paper, we aim to take advantage of $[-5,5]$ SMSD recoding of multiplier and dynamic negation of X-multiples, while reducing the number of XOR gates via generating $[-6,6]$ SMSD pre-computed X-multiples (i.e. just one XOR gate per 4-bit digit). Other contributions of this paper are highlighted below.

Starting the PPR with 16 partial products: An especial on the fly augmentation of two middle SMSD digits, leads to reducing the depth of partial product matrix by 1, such that the PPR starts with 16 operands right at the end of PPG, with no delay penalty for the latter.

Special 4-in-1 SMSD adder with TCSD sum: To avoid the challenging addition of SMSD IPPs, we design a novel carry-free adder that represents the sum of two $[-6, 6]$ SMSD operands in $[-7, 7]$ two's complement SD (TCSD) format, where one unified adder is utilized for all the four possible sign combinations.

Improved TCSD addition: The rest of the reduction process uses special TCSD adders that are actually an improved version of the fast TCSD adder of [5]. Such 2:1 reduction promotes the VLSI regularity of the PPR circuit, especially for $n = 16$ (i.e., the recommended operand size of IEEE 754-2008 [12]).

Augmenting the final redundant to non-redundant conversion with the last PPR level: The last PPR level would normally lead to TCSD product, which should be converted to BCD. However, to gain more speed and reduce costs, we device a special hybrid decimal adder, with two TCSD inputs and a BCD output.

The rest of this work is presented as follows. In Section II, as background coverage, we briefly study the previous relevant works, and discuss the corresponding IPP encodings and reduction cells. Regarding the proposed multiplier, SMSD recoding of digits of multiplier Y and pre-computed X-multiples, its PPR and final product computation are discussed in Section III. Analytical and synthesis results are presented in Section IV, where performance comparison with best previous works is also provided. We finally conclude in Section V.

II. Background

In this section, we briefly study several previous relevant works, via compiling their PPG and PPR characteristics in Table I. The column acronyms MR, ME, DN, #OP, PPDS, and PPDE stand for multiplier recoding, multiple encoding, dynamic negation of IPPs, number of operands to be added (i.e., number of originally generated nonredundant decimal numbers, or signed digit (SD) partial products), partial product digit set, and partial product digit encoding, respectively. In the same table, Svoboda [13] refers to the encoding of a digit $d \in [0, 6]$ ($[-6, -0]$) by 5-bit binary number $3d(31 - 3d)$. Some other works on decimal multiplication with floating-point operands [14-17], specific designs for FPGA (e.g., [18-19]), or digit by digit iterative approach (e.g., [9]), are not listed in Table I, since they are based on one of the tabulated works, or use embedded FPGA components, which are out of the scope of this work.

Some more details for those reference works that have been reportedly implemented, for $n = 16$, are:

- [20]: Sequential multipliers, with fast carry-free X-multiple generation, where in the process of partial product accumulation, the PPDS and PPDE transform from $[0, 18]$ to $[0, 10]$ and double-BCD to BCD carry-save (CS), respectively.
- [3]: As above, but the PPDS and PPDE transform to $[0, 15]$ and (8,4,2,1), respectively.
- [21]: Parallel multipliers, with fast carry-free X-multiple generation, where in PPR, the PPDS and PPDE transform to $[0, 15]$ and (8,4,2,1), respectively. Number of reduction levels is 5.
- [22]: As above, except that PPDS and PPDE transform to $[0, 10]$ and BCD CS, respectively. Number of reduction levels is 6.
- [2]: Parallel multiplier, with fast carry-free X-multiple generation, 4:1 and 2:1 multiplexing, where PPDS and PPDE will later transform to $[0, 10]$ and BCD CS, respectively. Number of reduction levels is 6.
- [6]: Two schemes are offered in this work, where both use 3:2 reduction and especial "$\times$ 2" correction cells.
  - Radix-5: Parallel multiplier, fast carry-free X-multiple generation, 4:1 multiplexing of $(\pm 2X, \pm X)$, and 2:1 multiplexing of $(10X, 5X)$. Number of reduction levels is 8.
  - Radix-10: Parallel multiplier, $[-5, 5]$ SMSD recoding of multiplier, slow carry-propagating $3X$ generation, 5:1 multiplexing with dynamic negation. Number of reduction levels is 6.
- [10]: Sequential multiplier, slow PPG via BCD-to-$[-5,5]$ SMSD recoding of multiplier's and multiplicand's digits, followed by digit-by-digit multiplication leading to $[-6, 6]$ PPDS with slow partial product accumulation via Svoboda adder [13].
- [7]: Parallel multiplier, $[-5, 5]$ SMSD recoding of multiplier, fast carry-free X-multiple generation via redundant representation of multiples including $3X$, 5:1 multiplexing with dynamic negation. Reduction is accomplished in two stages. One is 17:8 that includes three levels of CS adder (CSA) and a 4-bit adder. The other (i.e., 8:2) uses two levels of (4; 2) compressors and a 5-bit adder. Both stages conclude with some correction logic.
- [4]: As above, except for reduction, where IPPs are represented as $[0, 15]$ radix-10 numbers. Binary 4:2, and 3:2 reductions are used with due decimal corrections.

TABLE I
SUMMARY OF PPG AND PPR CHARACTERISTICS.

	Reference	MR	Pre-computed multiples	ME	DN	#OP	PPDS	PPDE
1	[11]	None	$\{0,1,\dots,9\} \times X$	BCD	No	$2n$	$[0, 9]$	BCD
2	[3], [20], [21]		$\{0,1,2,4,5\} \times X$				$[0, 18]$	Double BCD
3	[22]		$\{0,1,2,5,8,9\} \times X$					
4	[2]		$\{0, \pm 1, \pm 2,5,10\} \times X$					
5	[6] Radix-5			4,2,2,1				Double 4,2,2,1
6	[10]	$[-5, 5]$	None	$[-5, 5]$		$n + 1$	$[-6,6]$	Svoboda
7	[6] Radix-10		$\{0,1,2,3,4,5\} \times X$	4,2,2,1	Yes		$[0, 9]$	4,2,2,1
8	[7]			$[-8,8]$			$[-8,8]$	-8,4,2,1,1
9	[4]			$[-3,12]$ Excess-3			$[0, 15]$	8,4,2,1

In [4], [6], and [7] dynamic negation of pre-computed X-multiples reduces their selection cost at the penalty of one XOR gate per each bit of the selected positive multiple. This negation cost is replicated n times for parallel $n \times n$ multiplication. Moreover, the n inserted 1s for ten's complementation in [6], and $n \times (n+1)$ 1s for digit wise 2's complementation in [7], have a negative impact on area and power saving. The same is true for the correction constant, and more complex recoding due to zero handling, for $[0, 15]$ partial products in [4]. One way to save these costs, as we do in Section III, is to generate the signed digit pre-computed X-multiples with sign magnitude format, so as to reduce the XOR gates to one per digit (roughly 75% savings in the number of negating XOR gates) and remove the aforementioned negative impacts. However, besides slowing down the PPG to some extent (e.g., in comparison to radix-5 implementation of [6]), new problems are introduced in PPR, which is to be explained and solved in the next section, where we also reduce the depth of IPP matrix to $n = 16$, effectively prior to termination of PPG.

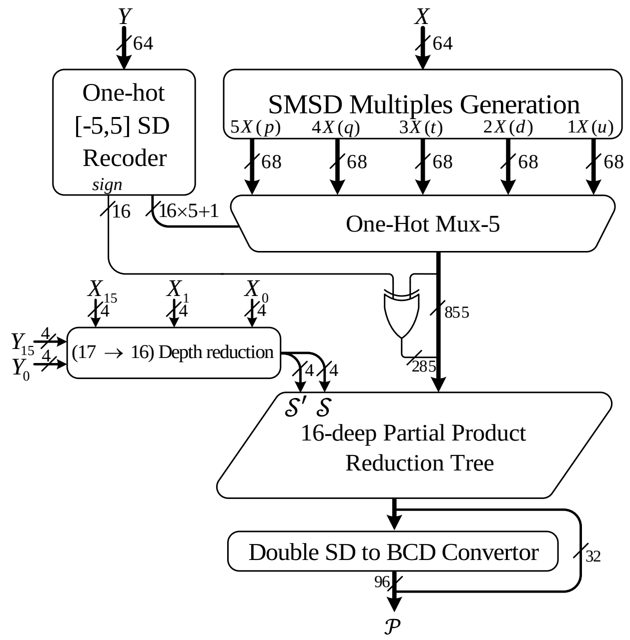
III. Decimal IPPs with Sign-magnitude representation of signed digits

Decimal signed digits in $[-\alpha, \alpha]$ ($\alpha \leq 7$) are usually encoded with minimal 4-bit signed numbers. For example, consider $\alpha = 5$ in [10] and $\alpha = 7$ in [23] with sign magnitude and 2's complement representations, respectively. The latter is suitable for basic arithmetic operations, except for negation, which is best performed on sign magnitude format.

In this section, we propose a decimal multiplication scheme with the following characteristics that are in the same line as those of the designs listed in Table I.

- $[-5, 5]$ SMSD recoding of multiplier's digits
- $\{0,1,2,3,4,5\} \times X$ pre-computed multiples
- 4-bit $[-6, 6]$ SMSD encoding of pre-computed multiples
- Dynamic negation of multiples with only one XOR per digit (i.e., per 4 bits)
- n (instead of $n+1$) operands to be added for $n \times n$ multiplication
- Unified SMSD+SMSD⟶TCSD adder for all four input sign combinations
- $[-7, 7]$ TCSD representation for accumulated partial products
- Early start of redundant to BCD conversion
- Augmenting last PPR level with final conversion to BCD

Fig. 1 depicts the general architecture of the proposed 16×16 multiplication $\mathcal{P} = X \times Y$, where details of each building block will be explained later. In particular, in the top three blocks, the multiplier's digits are recoded to n one-hot $[-5, 5]$ SMSDs (i.e., one sign, and 5 magnitude bits), augmented with a 10^n-weighted carry bit. The multiples $[0, 5] \times X$ are pre-computed as n $[-6, 6]$ SMSDs and a 10^n-weighted $[-5, 4]$ SMSD. Each SMSD contains a sign bit s and 3-bit magnitude. The negative multiples $[1, 5] \times (-X)$ are achieved via dynamic sign inversion of multiples $[1, 5] \times X$, at the cost of only one XOR gate per digit.



Y
64
X
64
One-hot [-5,5] SD Recoder
sign
SMSD Multiples Generation
5X (p)
4X (q)
3X (t)
2X (d)
1X (u)
68
68
68
68
68
16
16×5+1
One-Hot Mux-5
855
285
X15
X1
X0
4
4
4
Y15
Y0
4
4
(17 → 16) Depth reduction
4
4
S′ S
16-deep Partial Product Reduction Tree
Double SD to BCD Convertor
32
96
𝒫

Fig. 1. Overall block diagram view of the proposed multiplier.

3.1. Recoding of multiplier's digits

Original BCD digits of multiplier require $[0, 9] \times X$ pre-computed multiples, which include hard multiples $\{3, 6, 7, 9\} \times X$ that unlike $\{2, 4, 5, 8\} \times X$ are not derivable without carry propagation. On the other hand, BCD-to-redundant $[-5, 5]$ SMSD recoding of multiplier's digits with dynamic negation of IPPs reduces the required X-multiples to $[0, 5] \times X$ that include only one hard multiple (i.e., $3X$). However, this recoding produces a carry as the $(n+1)^{\text{th}}$ digit of multiplier, which increases the number of IPPs by 1. This is especially not desirable for $n = 16$ (i.e., the recommended IEEE754-2008 word size for decimal operands [12]). The reason is that it may increase the number of 2:1 PPR levels by 1, which can be avoided as will be dealt with in Section 3.3.

The one-hot recoding input/output expressions are given by Eqn. set 1, where $Y_i = v_3 v_2 v_1 v_0$, and $Y_{i-1} = w_3 w_2 w_1 w_0$ represent two consecutive digits of BCD multiplier, ω indicates whether $Y_{i-1} \geq 5$, $s_{v'}$ is the sign of target code, and v'_1-v'_5 are one-hot signals corresponding to absolute values of recoded multiplier's digit Y'_i (i.e., 1-5), whose decimal weight is equal to that of Y_i. More derivation details can be found in [4], [6], [7], and [10].

$$\boxed{\begin{aligned}
&\omega = w_3 \vee w_2(w_1 \vee w_0),\\
&v'_1 = \overline{v_2 \vee v_1}(\omega \oplus v_0),\\
&v'_2 = \omega v_0\left(\overline{v_3 \vee v_2 \vee v_1} \vee v_2 v_1\right) \vee \overline{\omega \vee v_0}(v_3 \vee \overline{v_2} v_1),\\
&v'_3 = v_1(\omega \oplus v_0),\\
&v'_4 = \overline{\omega \vee v_0} v_2 \vee \omega v_0 (v_2 \oplus v_1),\\
&v'_5 = v_2 \overline{v_1}(\omega \oplus v_0),\\
&s_{v'} = v_3 \overline{\omega v_0} \vee v_2(v_1 \vee v_0)
\end{aligned}} \qquad (1)$$

3.2. Pre-computed multiples

We need to generate $\{0,1,2,3,4,5\} \times X$, where X is a BCD multiplicand. The only hard multiple $3X$ can be generated in carry-free manner, if it is represented via a redundant digit set [5], [7]. Therefore, for uniformity sake in PPR, we generate all the required multiples in the same signed digit number system.

Let $X_i = b_3b_2b_1b_0$ and $X_{i-1} = a_3a_2a_1a_0$ denote two consecutive BCD digits of X, $3X_i = 10H_i + L_i$, and $3X_{i-1} = 10H_{i-1} + L_{i-1}$, where $0 \le 3X_i, 3X_{i-1} \le 27$, $H_i, H_{i-1} \in \{0,1,2\}$, and $L_i, L_{i-1} \in [0,9]$ (e.g., $X_iX_{i-1} = 59$ leads to $H_i = 1$, $H_{i-1} = 2$, $L_i = 5$, and $L_{i-1} = 7$). In cases that $L_i, L_{i-1} \ge 4$ (as in the latter example), we recode the 2-digit BCD number $H_i\ L_i$ to $H'_i\ L'_i$, and $H_{i-1}\ L_{i-1}$ to $H'_{i-1}\ L'_{i-1}$ based on Eqn. set 2, which leads to $H'_i, H'_{i-1} \in [0,3]$ and $L'_i, L'_{i-1} \in [-6,3]$.

$$\boxed{\begin{array}{l} L'_i = L_i - 10, H'_i = H_i + 1, \\ L'_{i-1} = L_{i-1} - 10, H'_{i-1} = H_{i-1} + 1 \end{array}} \quad (2)$$

For example, the new values per the above example (i.e., $X_iX_{i-1} = 59$) are $H_i = 2$, $H_{i-1} = 3$, $L_i = -5$, and $L_{i-1} = -3$.

Fig. 2 depicts the weighted organization of source, intermediate, and target digits of the above recoding, where $T_i = L'_i + H'_{i-1} \in [-6,3] + [0,3] = [-6,6]$. It is easy to verify that similar recoding can be applied to other multiples to be represented with the same digit set ($[-6,6]$ SMSD).

Decimal position		$i+1$	i	$i-1$
X	$[0,9]$	X_{i+1}	X_i	X_{i-1}
$3X$	$[0,9]$		L_i	L_{i-1}
	$[0,2]$	H_i	H_{i-1}	
$3X$	$[-6,3]$		L'_i	L'_{i-1}
	$[0,3]$	H'_i	H'_{i-1}	
$3X$	$[-6,6]$		T_i	

Fig. 2. Two consecutive digits of X, $3X$ (BCD), and $3X$ ($[-6,6]$SMSD).

The corresponding logical expressions for SMSD multiples of the multiplicand X (i.e., $1X(u)$, $2X(d)$, $3X(t)$, $4X(q)$, and $5X(p)$) can be derived in terms of bits of BCD digits a (in position i) and b (in position $i-1$). For example, that of $3X(t)$, is given by Eqn. set 3, and the rest can be found in the Appendix. Note that such multiples are represented with at most one extra digit (i.e., total of 68 bits), since the most significant digit of the generated multiple is at most 4 (due to $5 \times 9 = 45$), which remains 4 within the BCD-to-SMSD conversion.

$$\boxed{\begin{array}{l} s_t = \overline{a}_3\overline{a}_2(\overline{a}_1\overline{b}_2b_1 \vee b_1\overline{b}_0) \vee b_1\overline{b}_0(\overline{b}_2 \vee \overline{a}_3\overline{a}_1\overline{a}_0) \vee \\ \qquad b_3(\overline{a}_3 \vee \overline{b}_0) \vee b_2\overline{b}_1b_0, \\ t_2 = \overline{b}_1b_0(\overline{b}_3\overline{b}_2(a_3 \vee a_2) \vee \overline{a}_3b_2(\overline{a}_2 \vee \overline{a}_1\overline{a}_0) \vee \overline{a}_2a_1\overline{b}_3) \vee \\ \qquad b_2\overline{b}_1\overline{b}_0(a_3 \vee a_2a_1 \vee a_2a_0) \vee \\ \qquad \overline{a}_3\overline{b}_0(b_3 \vee \overline{a}_2\overline{a}_1\overline{b}_2b_1)a_3b_2b_1b_0, \\ t_1 = \overline{b}_2\overline{b}_1(a_2\overline{b}_3(a_1 \vee a_0) \vee b_1(a_2 \vee a_1) \vee a_3\overline{b}_1) \vee \\ \qquad \overline{a}_3\begin{pmatrix} \overline{a}_2(b_3b_0 \vee b_2\overline{b}_1\overline{b}_0) \vee \overline{a}_2\overline{a}_1(b_2\overline{b}_0 \vee \overline{b}_2\overline{b}_1b_0) \vee \\ \overline{a}_1\overline{a}_0(b_3b_0 \vee b_2\overline{b}_1\overline{b}_0) \end{pmatrix} \vee \\ \qquad b_2b_0(a_2(a_1 \vee a_0) \vee b_1(a_2 \vee a_1)) \vee \\ \qquad a_3\overline{b}_3b_0(\overline{b}_2 \vee \overline{b}_1) \vee \overline{a}_2\overline{a}_1b_3\overline{b}_0, \\ t_0 = b_0(a_2(a_1 \vee a_0) \vee \overline{a}_3\overline{a}_2\overline{a}_1) \vee \\ \qquad \overline{b}_0(a_3 \vee \overline{a}_2a_1 \vee a_2\overline{a}_1\overline{a}_0) \end{array}} \quad (3)$$

3.3. Partial product generation

Fig. 3 depicts the PPG, and the normal organization of IPPs of such $n \times (n+1)$ multiplication for $n = 4$. The tick bars represent BCD digits of the multiplicand. Depth of the deepest column of IPP matrix (i.e., 10^n-weighted position) is $(n+1)$, where all digits belong to $[-6,6]$, except the top and bottom ones (in gray) that belong to $[-5,4]$ and $[-6,3]$, respectively.

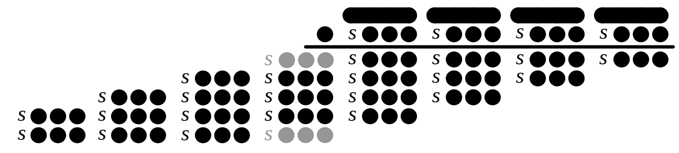


Fig. 3. Normal organization of intermediate partial products.

We reduce the matrix depth to n (e.g., $5 \to 4$ for $n = 4$, and $17 \to 16$ for $n = 16$), with no delay between the termination of PPG and start of PPR. Here is how it works: we compute sum of the two gray digits (see Fig. 3) independent of (and in parallel to) normal PPG, as follows. If $Y_{n-1} \le 4$, the value of 10^n-weighted carry of recoded multiplier is zero, so the bottom gray digit has to be zero. Therefore no addition is required. For $Y_{n-1} > 4$, let H denote the most significant digit of $X_{n-1} \times Y'_0$ (e.g., the top gray digit in Fig. 3) where X_{n-1} and Y'_0 represent the most significant BCD digit of multiplicand and the least significant recoded digit of multiplier, respectively. We extract H as ten one-hot signals via an 8-input logic (see the rightmost box in Fig. 4).

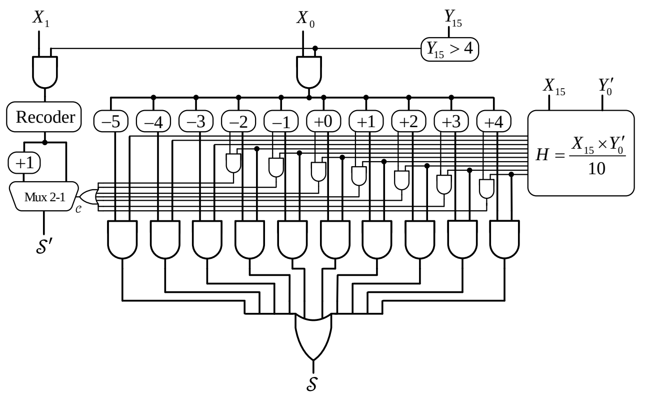


Fig. 4. The required circuit for ($17 \to 16$) depth reduction.

The least significant BCD digit of multiplicand (i.e., X_0), as is illustrated in the rest of Fig. 4, is added to constants in $[-5,4]$. This leads to the desired sum digit $\mathcal{S}$ in 10^n-weighted position (in place of two gray digits of Fig. 3) and a carry bit c to be added to the 10^{n+1}-weighted digit next to bottom gray digit to result in $\mathcal{S}'$ ($\mathcal{S}$ and $\mathcal{S}'$ are also distinguished by white triangles Δ in Fig. 5 in Section 3.4). This digit, as shown in the leftmost part of Fig. 4, is obtained by directly recoding the 10-weighted digit of multiplicand (i.e., X_1).

3.4. Partial product reduction

The overall PPR for $n = 16$ is illustrated by Fig. 5, where a bar, triangle, square, and diamond represent a BCD, $[-6,6]$ SMSD, $[-7,7]$ TCSD, and binary signed digit (BSD), respectively. The choice of SMSD representation for the first level IPPs, while facilitates the PPG, bears no extra complexity for PPR, since all reduction levels use TCSD adders, except for the first one that requires a special SMSD+SMSD-to-TCSD adder. However, as will be shown at the end of Section 3.4.1 this adder is not more complex than a simple TCSD adder.

The red shaded SMSD in Level II of Fig. 5 is directly converted to BCD. Similar direct conversions are in order for the red shaded digits (TCSDs, however) in the subsequent Levels III and IV.

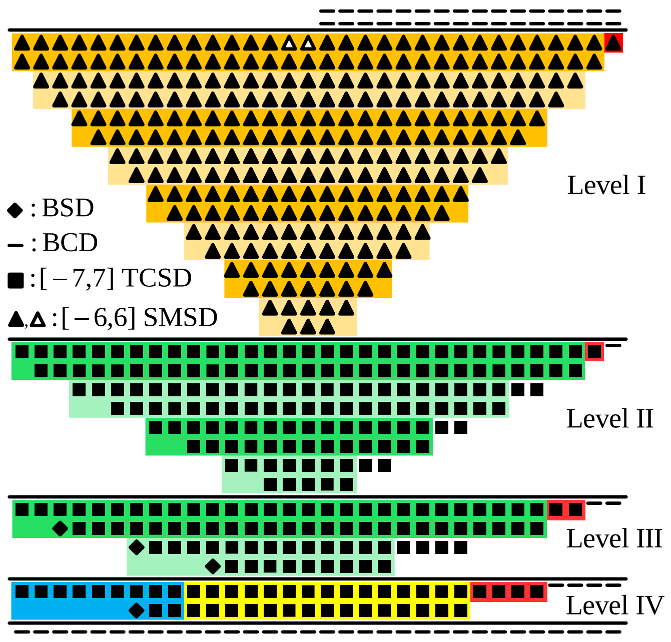


Fig. 5. The overall view of 16×16 digit multiplier.

3.4.1) *Special 4-in-1 SMSD adder*

A digit slice of the aforementioned SMSD+SMSD-to-TCSD adder, for four different cases corresponding to all possible combinations of the input signs, are depicted by Fig. 6 (**a**, **b**, **c**, **d**) in dot-notation representation. The black and white dots represent posibits and negabits (a posibit is a normal bit whose arithmetic value equals its logical status, and the arithmetic value of a negabit with logical status x equals $x-1$ [24]. The sum of two $[-6,6]$ SMSD digits (e.g., $P = s_p p_2 p_1 p_0$ and $Q = s_q q_2 q_1 q_0$), and a signed carry-in (e.g., C_{in}) is produced as one $[-7,7]$ TCSD digit (e.g., $S = s_3 s_2 s_1 s_0$), and a signed carry-out (e.g., C_{out}). This is a 2-stage process. In the stage I, the sign bits are applied to the magnitudes, such that a negative sign changes the polarity of magnitude posibits to negabits and inverts their logical states. Subsequently, in the same stage, the bit collection U is decomposed, and the bit collection V is recoded. In the second stage, however, as will be explained shortly, only one 4-bit adder takes care of all the four cases, which explains the rationale for designation of the adder.

- **Decomposition of $\boldsymbol{U}$**: Following the partitioning technique of [5], we show that the bit-collection $U = (-1)^{s_p}(2p_2 + p_1) + (-1)^{s_q}(2q_2)$, can be decomposed to Z and C_{out} bit collections, such that $2U = 2Z + 10C_{out}$. Table II contains the details of such decomposition for the four possible (s_p, s_q) combinations, where it is shown that $C_{out} \in [-1, 1]$ and Z can be extracted from U values. Furthermore, the BSD signed carry C_{out} is represented as a posibit/negabit pair (c'_{out}, c''_{out}), and to represent the Z values in each case a 3-bit encoding that covers the corresponding range is proposed. For example, in case of $s_p = +$ and $s_q = -$, the arithmetic range of $Z (= 4z_3 + 2(z_2 - 1) + z_1)$, is $[-2, 5]$. This range covers that of $Z = \frac{(2U - 10C_{out})}{2} \in [-2, 3]$ (i.e., $Z \in \{4, 5\}$ never occurs), which makes the decomposition valid. The required logical expressions for the bits of C_{out} and Z that are derived via simple 5-input truth tables are presented in Eqn. set 4.

$$\boxed{\begin{aligned}
&c'_{out} = \overline{s_p \vee s_q}(p_2 \vee q_2),\ c''_{out} = s_p s_q (p_2 \vee q_2)\\
&z_1 = s_p \overline{s_q}\ \overline{q_1} \vee \overline{s_p} p_1 \left(s_q \vee p_2 \vee q_2\right) \vee s_q p_1 (p_2 \vee q_2) \vee\\
&\qquad \overline{p_2 \vee q_2 \vee p_1}\ (s_p \vee\ \overline{s_q}),\\
&z_2 = p_2 q_2 \left(\overline{s_p} \vee \overline{s_q} \vee \overline{p_1}\right) \vee \overline{p_2 \vee q_2}\ (\overline{s_p}\ \overline{p_1} \vee s_p \oplus s_q) \vee\\
&\qquad s_p s_q p_1 p_2 \oplus q_2,\\
&z_3 = s_q p_2 \overline{q_2}\ \overline{s_p p_1} \vee s_p \overline{p_2} q_2 \overline{s_q p_1} \vee \overline{s_p \vee s_q}\ \overline{p_2 \vee q_2} p_1
\end{aligned}} \quad (4)$$

- **Recoding of $\boldsymbol{V}$ to $\boldsymbol{V'}$**: The V bit collection is described as $V = (-1)^{s_p}(p_0) + (-1)^{s_q}(2q_1 + q_0)$, which is to be recoded to V' bit collection with the same arithmetic value. Note that the bit polarities in V' are different in the four cases of Fig. 6. For example, in Fig. 6b, the arithmetic value of V equals to $-2q_1 + p_0 - q_0 \in [-3, 1]$, while that of V' is $4v_2 + 2(v_1 - 1) + (v_0 - 1) \in [-3, 4]$ that covers the original range $[-3, 1]$. These recodings can be done via a circuit that is described by Eqn. set 5 (also derived via a 5-input simple truth table).

$$\boxed{\begin{aligned}
&v_0 = \overline{p_0 \oplus q_0},\\
&v_1 = (q_1 \oplus q_0)\overline{s_q \oplus p_0} \vee (s_q \oplus p_0)\overline{s_p \oplus q_1},\\
&v_2 = s_q \overline{q_1}(p_0 \overline{q_0} \vee s_p \overline{p_0}) \vee \overline{s_q} q_1 (\overline{p_0} q_0 \vee \overline{s_p} p_0)
\end{aligned}} \quad (5)$$

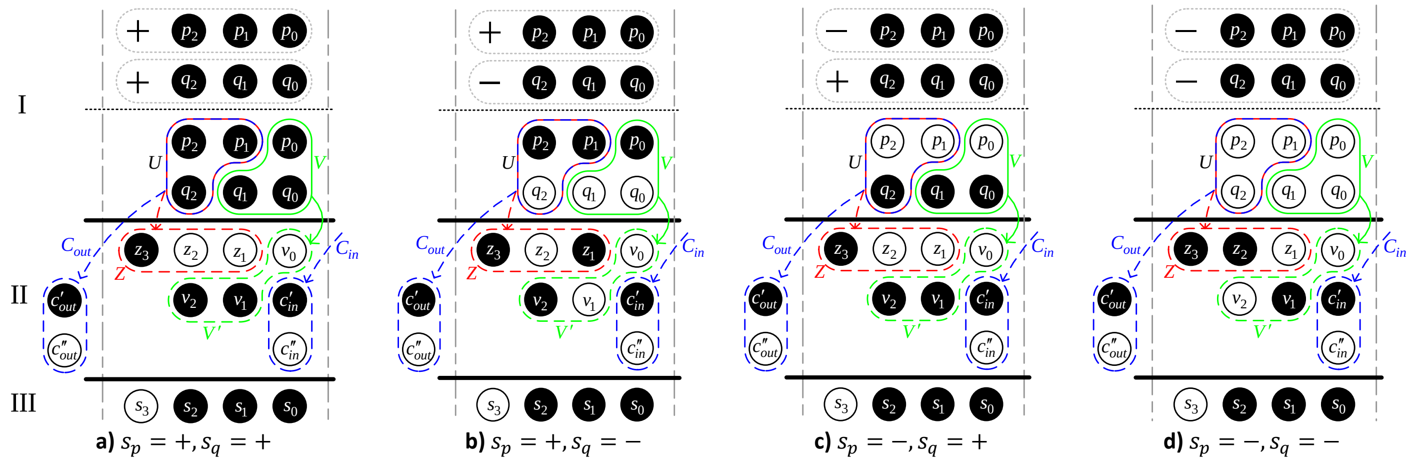


Fig. 6. A digit slice of SMSD+SMSD-to-TCSD adder for four sign combinations.

TABLE II
VALUE OF U, V', Z, C_{out}, AND FINAL RESULT IN FOUR CASES $s_p\, s_q = \{++, --, +-, -+\}$.

Sub Fig. 6	s_p s_q	$P + Q = 2U + V$	U	C_{out}	$Z = (2U - 10C_{out})/2$	$V' = V$	$2Z + V' + C_{in}$
a	$+$ $+$	$[0, 12]$	$[0, 5]$	$\{0, 1\}$	$[-3, 2]$ (●○○)	$[0,4]$ (●●○)	$[-7, 7]$ (○●●●)
b	$+$ $-$	$[-6, 6]$	$[-2, 3]$	0	$[-2, 3]$ (●○●)	$[-3,1]$ (●○○)	
c	$-$ $+$	$[-6, 6]$	$[-3, 2]$	0	$[-3, 2]$ (●○○)	$[-1,3]$ (●●○)	
d	$-$ $-$	$[-12, 0]$	$[-5, 0]$	$\{-1, 0\}$	$[-1, 3]$ (●●○)	$[-4,0]$ (○●○)	

- **The 4-in-1 design:** Other encodings are also possible for Z and V' values. For example, an alternative encoding for both $Z \in [-2, 3]$ and $V' \in [-3, 1]$ of Fig. 6b is ○●●$\in [-4, 3]$ that covers the latter two intervals. However, the proposed encodings (see Table II) are so chosen to allow for unified treatment of the bit collections that are obtained after the decomposition and recoding. That is a simplified 4-bit adder (see Fig. 7) can take care of all the four cases. This is actually possible via the standard full adders that are capable of handling all the 3-bit posibit/negabit collections of inputs [24]. Note that the normally required leftmost HA is reduced to an OR gate since no carry out is expected.

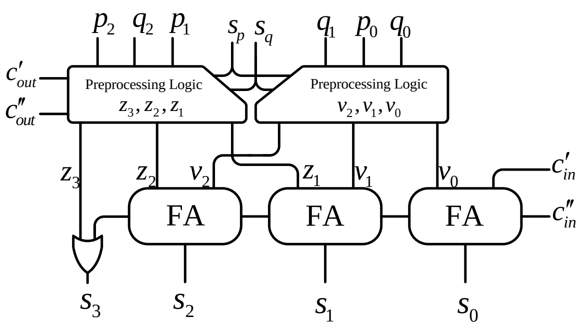

p_2 q_2 p_1 s_p s_q q_1 p_0 q_0
c'_{out}
c''_{out}
Preprocessing Logic z_3, z_2, z_1
Preprocessing Logic v_2, v_1, v_0
z_3 z_2 v_2 z_1 v_1 v_0
c'_{in}
c''_{in}
FA
FA
FA
s_3 s_2 s_1 s_0

Fig. 7. A digit slice of the 4-in-1 SMSD + SMSD → TCSD adder.

The aforementioned decomposition and recoding can be further justified by close examination of the content of Table II, where the range of $P + Q$ determines the possible values for C_{out}, which always lead to $S = 2Z + V' + C_{in} \in [-7, 7]$, as is shown in the rightmost column.

The (c'_{in}, c''_{in}) pair represents the incoming signed carry C_{in} from the less significant position. Representations of Z, V', and C_{in} are so determined as to lead to two's complement representation for S, in all the four cases (see below for more explanations, and the following numerical example).

Example 1 (Fig. 6 by numerical values): Fig. 8 describes a numerical example, where two SMSDs $P = s_p101$ ($|P| = 5$) and $Q = s_q100$ ($|Q| = 4$) are added. This figure mimics Fig. 6 with numerical values, where signs (i.e., s_p and s_q) are explicitly shown as was the case in Fig. 6, and negabits are inversely encoded as $1^-(0^-)$, which represent arithmetic value $0(-1)$. The incoming signed carry $C_{in} = 0$ is represented by the posibit $c'_{in} = 0$ and inversely encoded negabit $c''_{in} = 1^-$. Therefore, the FA in position 0 receives two negabits and one posibit, and produces a posibit sum 1 and a negabit carry 0^-, such that $2 \times (-1) + 1 = -1$, as there was only one arithmetically nonzero input 0^- (i.e., -1).

The 4-in-1 adder is slightly more efficient than $[-7, 7]$ TCSD adder (i.e., less latency with no area overhead), as can be verified by inspecting Eqn. sets 4-5, for the preprocessing logic boxes in 4-in-1 adder and that of TCSD adder (i.e., Eqn. set 6 in Section 3.4.2).

3.4.2) TCSD adder

The TCSD adder, which is required for the remaining $(\lceil \log_2 n \rceil - 2)$ subsequent reduction levels (i.e., Levels II and III in Fig. 5), is an improved version of that of [5]. The required architecture is the same as in Fig. 7, except for the preprocessing boxes, where the required logical expressions are described in Eqn. set 6. Also Fig. 9 depicts one digit slice of this adder.

$$\boxed{\begin{aligned} & c'_{out} = \overline{p_3}\,\overline{q_3},\ c''_{out} = p_3q_3\overline{p_2q_2p_1} \vee \overline{p_2}\,\overline{q_2}(p_1 \vee \overline{p_3}\,\overline{q_3}) \\ & v_0 = \overline{p_0 \oplus q_0},\ v_1 = q_1 \oplus (p_0 \vee q_0),\ v_2 = q_1(p_0 \vee q_0), \\ & z_1 = (p_2 \vee q_2)(p_3\overline{q_3}\,\overline{p_1} \vee \overline{p_3}(q_3 \oplus p_1)) \vee \overline{p_3}\,\overline{q_3}\,\overline{p_2}\,\overline{q_2}\,\overline{p_1} \vee \\ & \qquad p_3q_3p_1\overline{p_2q_2}, \\ & z_2 = (p_2 \oplus q_2)(p_3(q_3 \vee p_1) \vee q_3p_1) \vee p_2q_2\overline{p_1}\,\overline{p_3q_3} \vee \\ & \qquad \overline{p_3}\,\overline{q_3}\,(p_2q_2 \vee \overline{p_2}\,\overline{q_2}\,\overline{p_1}), \\ & z_3 = \overline{p_3}\,\overline{q_3}\,\overline{p_2}\,\overline{q_2}\,p_1 \vee p_3q_3p_2q_2\overline{p_1} \vee \\ & \qquad (p_3 \oplus q_3)(p_2q_2p_1 \vee \overline{p_2}\,\overline{q_2}\,\overline{p_1}) \end{aligned}} \quad (6)$$

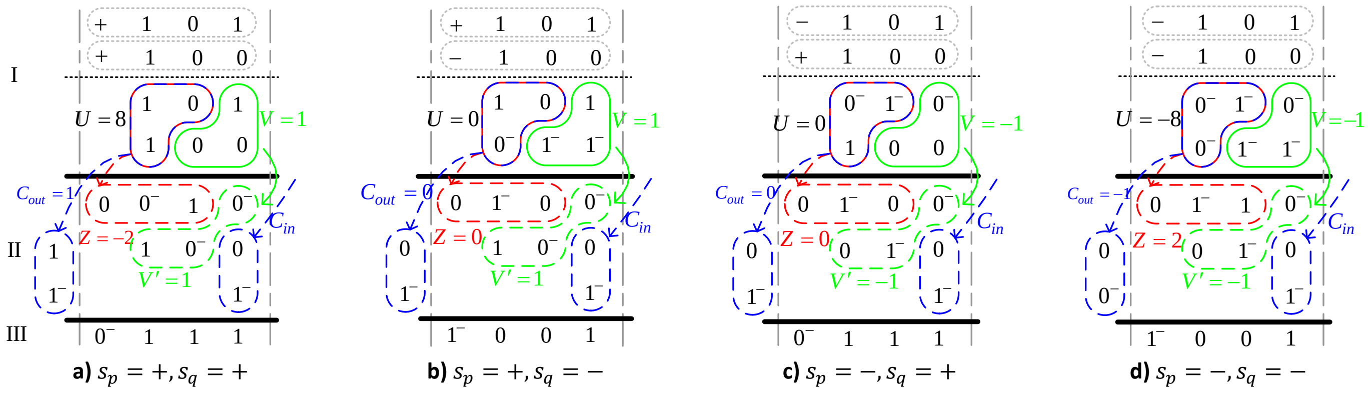


Fig. 8. Numerical example with $|p| = 5$ and $|q| = 4$.

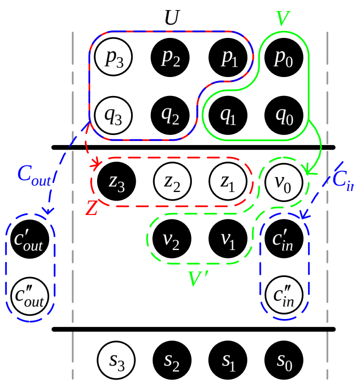


Fig. 9. TCSD adder.

These 2:1 TCSD reductions lead to two $[-7,7]$ TCSD accumulated partial products that are summed up with a special adder that directly produces the final BCD product, whose details are explained in Section 3.5.

3.5. Final product computation

Recall that the Level IV of Fig. 5 contains two $[-7,7]$ TCSD IPPs. To get at the final product the straightforward method calls for a final 2:1 reduction level that leads to a $2n$-TCSD product $D_{2n-1} \dots D_0$. This is then to be converted to the equivalent BCD product $\mathcal{P}_{2n-1} \dots \mathcal{P}_0$, which could be done via the recurrence of Eqn. set 7, where $D_i \in [-7,7]$, $b_i \in [-1,0]$, and $\mathcal{P}_i \in [0,9]$, for $0 \le i \le 2n-1$.

$$\boxed{\begin{array}{l} b_0 = 0\,, \\ W_i = D_i + b_i,\, (b_{i+1}, \mathcal{P}_i) = \begin{cases} (0, W_i) & \text{if } W_i \ge 0 \\ (-1, 10 + W_i) & \text{if } W_i < 0 \end{cases} \end{array}} \quad (7)$$

To speed up the latter two steps (i.e., 2:1 reduction and TCSD-to-BCD conversion), the actual BCD product generation of Fig. 5 uses a more efficient method to be described below. The final 2:1 reduction level that is required for positions 8 to 22 and the subsequent TCSD-to-BCD conversion can be actually augmented as a TCSD + TCSD addition with BCD result, which will be explained in the Section 3.5.2, below. However, the product digits for positions 0-7 can be directly converted to BCD on the fly as is discussed in Section 3.5.1. Finally, the combined reduction and conversion in the remaining most significant positions are described in Section 3.5.3. A similar 3-part final product generation, for binary multiplication is undertaken in [25].

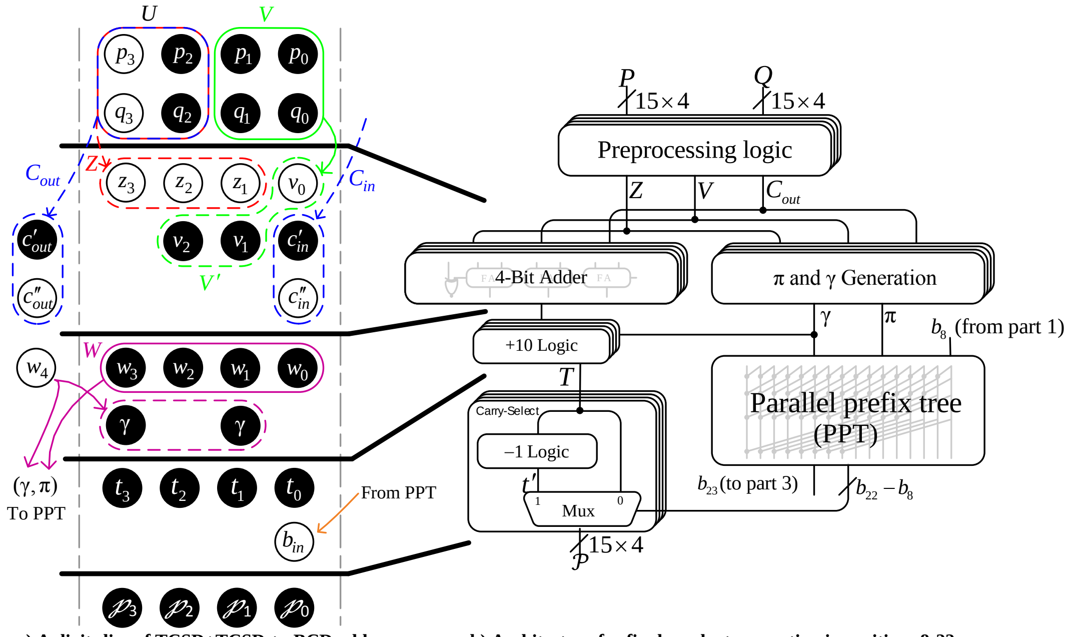
3.5.1) Positions 0-7

Following some previous works on decimal multiplier designs (e.g., [22], [6], [7]), we take advantage of different arrival times of the product digits for position 0-7 (red-shaded in Fig. 5). The least significant product digit is obtained in Level I as an SMSD digit, which is directly converted to BCD via Eqn. set 7. The next product digit that is available at Level II, as a TCSD, is likewise converted. So is the case for TCSDs D_3-D_2 and D_7-D_4 that are delivered in Levels III and IV, respectively.

3.5.2) Positions 8-22

There are two TCSD digits per positions 8-25. We don't apply another PPR level (i.e., TCSD+TCSD-to-TCSD conversion, as in Fig. 9). Instead, we can think of a TCSD+TCSD-to-BCD converter that can be realized with the help of a parallel prefix adder. However, the reason that we discuss the 15 positions 8-22 (distinguished by yellow shading in Fig. 5) separately in this section is that they together with the borrow-in signal b_8 contribute to a fully utilized 16-bit parallel prefix tree.

A digit slice of the aforementioned converter is illustrated by Fig. 10a. The function of this addition scheme is similar to that of Fig. 9, except that the collective value of the eight bits due to C_{in}, Z, and V' variables (i.e., $W = w_4 w_3 w_2 w_1 w_0 = 4U + V + C_{in} - 10C_{out} = 2Z + V' + C_{in}$) belong to $[-9,7]$. Note that the decomposition of $4U$ to $(2Z + 10C_{out})$, as in Eqn. 8, is undertaken such that $Z \in [-4,0]$ is composed of only negabits. Each of these signals can be extracted by separate 4-input combinational logic.



a) A digit slice of TCSD+TCSD-to-BCD adder. **b) Architecture for final product generation in positions 8-22.**

Fig. 10. Final conversion in Part 2.

$$\boxed{\begin{array}{l} c'_{out} = \overline{p_3}\,\overline{q_3}(p_2 \vee q_2),\, c''_{out} = p_3q_3\overline{p_2q_2} \\ v_0 = p_0 \oplus q_0,\, v_1 = p_1 \oplus q_1 \oplus (p_0 \vee q_0), \\ v_2 = p_1q_1 \vee (p_1 \vee q_1)(p_0 \vee q_0), \\ z_1 = c'_{out} \vee c''_{out},\, z_2 = \overline{p_3q_3}p_2 \oplus q_2 \vee p_3q_3\overline{p_2}\,\overline{q_2}, \\ z_3 = \overline{p_2 \ \vee q_2}\, p_2 \oplus q_2 \vee p_3q_3p_2q_2 \end{array}} \quad (8)$$

Since we seek BCD product, we convert W digits to BCD digits T, via Eqn. 9, where w_4 is a weighted-16 negabit. The $-6\overline{w_4}$ operation can be replaced by $+10\overline{w_4}$. However, in case of $w_4 = 0$, a decimal borrow is carried over to the more significant decimal position that causes borrow propagation. To avoid such slow borrow propagation, we employ a parallel prefix borrow generator that uses decimal borrow propagate and generate signals $\pi = (W = 0)$ and $\gamma = (W < 0) = \overline{w_4}$, respectively. These borrow signals are generated via a 4-level Kogge-Stone (KS) [26] parallel prefix network with 15 input pairs (π,γ), and borrow-in $b_8$from Part 1 (i.e., out of position 7).

$$\boxed{T = \begin{cases} w_3w_2w_1w_0 & \text{if } w_4 = 1 \\ w_3w_2w_1w_0 - 6 & \text{if } w_4 = 0 \end{cases} = w_3w_2w_1w_0 - 6\overline{w_4}} \quad (9)$$

To avoid 4-bit borrow propagation within each $T = t_3t_2t_1t_0$ digit, we also concurrently compute $T' = T - 1$, where one of T or T' is to be selected by borrow b_{in} that yields the product digit $\mathcal{P} = \wp_3\wp_2\wp_1\wp_0$. Fig. 10b depicts the logical blocks that correspond to different stages of Fig. 10a.

3.5.3) Positions 23-31

The π and γ signals for decimal positions 23-25 are produced similar to those of Section 3.5.2. Regarding the positions 26-31, where there exists only one $[-7, 7]$ TCSD per position, γ is equal to the NOT of sign bit of the corresponding TCSD, and π can be derived as the NOR of all four bits (sign bit inverted). We devise a special 3-level compound KS-like parallel prefix network to generate all borrows $b^0(b^{-1})$ for decimal positions 24-31 that correspond to the cases where b_{23} is 0 (1). Fig. 11 depicts the required logic, where (Γ, Π) represent the group (generate, propagate) signals. These borrows are utilized to form two BCD products $\mathcal{P}_{31} \ldots \mathcal{P}_{24}\mathcal{P}_{23}$ and $\mathcal{P}_{31} \ldots \mathcal{P}_{24}\mathcal{P}_{23} - 1$ corresponding to $b^0_{31} \ldots b^0_{24}b_{23}$ and $b^{-1}_{31} \ldots b^{-1}_{24}b_{23}$, respectively, where one is selected by b_{23}.

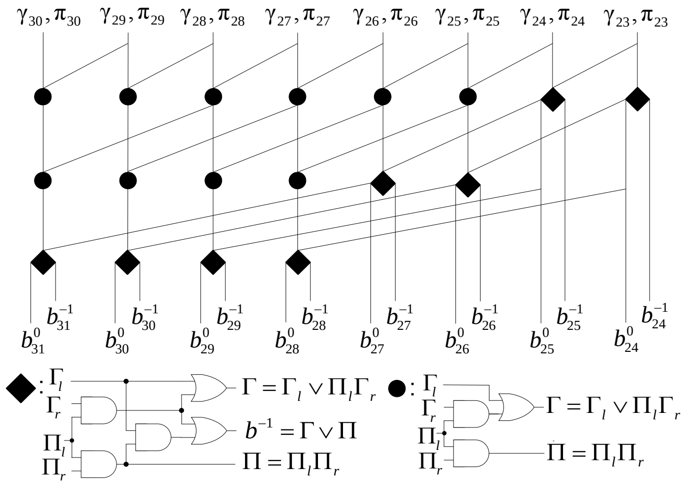


Fig. 11. The 3-level compound KS-like parallel prefix network.

Note that, given the one gate-level earlier availability of Π signals with respect to companion Γ signals, the special diamond node is so designed as to produce b^{-1} no later than b^0; of course with no delay overhead for the latter.

IV. Evaluation and comparisons

In this section, we provide analytical evaluation of latency of the proposed multiplier and those of [2], [21], [22], [6], [7], and [4]. These include all the previously reported parallel decimal multipliers, except for [27] that only provides synthesis results and no sufficient information to enable analytical evaluation. However, for more reliable results and fair comparison, we will provide, in Section 4.2, the synthesis-based figures of merit for all the aforementioned designs.

4.1. Analytical evaluation

Tables III-V contain delay measures of PPG, PPR, and final product computation (respectively), and their components, of the proposed design and those of [2], [21], [22], two of [6], [7] (based on the reevaluation in [28]), and the recent work of [4]. Also, the corresponding overall delay measures are compiled in Table VI.

We could not actually copy the analytical evaluation results of all the reference works, since the work of [2] provides only synthesis results. Those of [22] and [6] are in terms of FO4, where their underlying FO4 evaluation assumptions are not apparently the same and thus could not be followed in the evaluation of our design. Therefore, for fair comparisons, we preferred to derive the entries in rows 1-7 of Tables III-V directly from the design description of the corresponding articles, in the same way that we did with our design. These gate level evaluations are in terms of ΔG (i.e., delay of a 2-input simple gate), which is easily verifiable.

TABLE III
LATENCY COMPARISON OF PPG STAGE (ΔG).

	Reference	Components	Delay	Total	Ratio
1	[2]	$-2X$ generation	6	17	1.42
		Mux 4:1	3		
		BCD full adder [29]	8		
2	[22]	$8X$ generation	4	7	0.58
		Mux 3:1	3		
3	[21]	$4X$ generation: Δg	8	11	0.92
		Mux 3:1	3		
4	[6] Radix-5	BCD to 4,2,2,1 conversion	1	8	0.67
		$2X$ generation	4		
		Mux 4:1	3		
5	[6] Radix-10	$3X$ generation*	21	27	2.25
		Mux 5:1	4		
		Dynamic negation	2		
6	[7]	$3X$ generation	7	13	1.08
		Mux 5:1	4		
		Negation	2		
7	[4]	$4X$ genration	6	12	1.00
		Mux 5:1	4		
		Negation	2		
8	Proposed	$4X$ genration	8	12	1.00
		Mux 5:1	4		

*17Δg for $2X + X$ (16-digit BCD parallel prefix adder), and 4Δg for $2X$ generation

TABLE IV
LATENCY COMPARISON OF PPR STAGE (ΔG).

	Reference	Components		Delay	Total	Ratio
1	[2]	Six reduction levels with BCD full adder of [29]		6×8	48	1.37
2	[22]	Six reduction levels with BCD full adder of [29]		6×8	50	1.43
		Mux 2:1		2		
3	[21]	Two levels simplified OODS adder		2×10	56	1.6
		Three levels OODS adder		3×12		
4	[6] Radix-5	(8:4) counter		6	51	1.46
		Five levels full adder		$3 + 4 + 3 + 3 + 3$		
		Five correction "$\times$ 2" cells		5×5		
		4,2,2,1 to 5,4,2,1 conversion		4		
5	[6] Radix-10	(9:4) counter		7	39	1.11
		Four levels full adder		$3 + 4 + 3 + 3$		
		Three correction "$\times$ 2" cells		3×5		
		4,2,2,1 to 5,4,2,1 conversion		4		
6	[7]	L1	(5; 2) Compressor	8	$22 + 26$	1.37
			4-bit carry look ahead adder	5		
			XOR	2		
			Transfer logic	7		
		L2	$2 \times$ (4; 2) Compressor	2×6		
			4-bit carry look ahead adder	5		
			XOR	2		
			Transfer logic	7		
7	[4]	14-bit counter		23	45	1.29
		$\times$ 2 correction		10		
		Block A		12		
8	Proposed	SMSD to TCSD adder		11	35	1.00
		Two levels TCSD adder		2×12		

The 12 ΔG PPG critical delay path of our multiplier travels through $4X$ generation circuit, which takes 8 ΔG (see Eqn. set A3, in the Appendix), and crosses the 5:1 one-hot multiplexor (see Fig. 1) with 4 ΔG latency. The required dynamic negation is not within the critical delay path, since logical expressions for sign bits take at most 10 ΔG (see Appendix A) to complete.

TABLE V
LATENCY COMPARISON OF FINAL ADDITION STAGE (ΔG).

	Reference	Components	Delay	Total	Ratio
1	[2]	Decimal P & G generation	3	15	0.71
		32-digit parallel prefix tree	5×2		
		Mux 2:1	2		
2	[22]	Binary P & G generation	1	17	0.81
		104-bit parallel prefix tree	7×2		
		Mux 2:1	2		
3	[21]	4-bit P & G generation	2	14	0.67
		32-digit parallel prefix tree	5×2		
		Mux 2:1	2		
4	[6] Radix-5	Binary P & G generation	1	17	0.81
		128-bit parallel prefix tree	7×2		
		Mux 2:1	2		
5	[6] Radix-10	Binary P & G generation	1	17	0.81
		128-bit parallel prefix tree	7×2		
		Mux 2:1	2		
6	[7]	Decimal P & G generation	7	25	1.19
		Generation of C_{16}	6		
		Generation of C_{32}	8		
		Final conditional constant adder	4		
7	[4]	Binary P & G generation	1	17	0.81
		128-bit parallel prefix tree	7×2		
		Mux 2:1	2		
8	Proposed	Preprocessing logic (Eqn. Set 8)	4	21	1.00
		decimal propagate signal π	7		
		4-level parallel prefix tree	4×2		
		Mux 2:1	2		

Also the latency of (17→16) depth reduction logic (see Fig. 4) that operates in parallel with general PPG (see Fig. 1) is 12 ΔG.

The 35 ΔG PPR delay of the proposed multiplier equals the sum of latencies of following components: 11 ΔG (See Eqn. set 4 and 5) for first level of Fig.5 that reduces 16 SMSD IPPs to 8 TCSD IPPs, and 2×12 ΔG (See Eqn. set 6) for the next two levels.

Regarding the 21ΔG latency of the final stage of the proposed multiplier, recall Section 3.5, where BCD product computation takes 4ΔG for Eqn. set 8, 7ΔG for decimal propagate signal π, 8 ΔG for the 4-level Kogge-Stone parallel prefix tree, and 2ΔG for selecting sum or $sum - 1$.

As is evident from the complied ratios in Table VI, the proposed multiplier operates at least 9% faster than those of all the previous relevant works. In particular, our PPR latency is the least 10ΔG and 15ΔG less than the fastest previous multipliers in [22], and [4], respectively. Since it takes only three levels of fast 2:1 reduction, as it effectively starts with 16 partial products down to 2 partial products, which undergo the conversion to BCD without being reduced to the final redundant partial product. The latter augmentation is at the cost of 4ΔG prolongation in generating the final BCD product, but saves 12ΔG delay of double-TCSD-to TCSD reduction.

TABLE VI
OVERALL LATENCY COMPARISON OF EIGHT DESIGNS (ΔG).

	Reference	PPG	PPR	BCD product	Total	Ratio
1	[2]	17	48	15	80	1.18
2	[22]	7	50	17	74	1.09
3	[21]	11	56	14	81	1.19
4	[6]-R5	8	51	17	76	1.12
5	[6]-R10	27	39	17	83	1.22
6	[7]	13	48	25	86	1.26
7	[4]	12	45	17	74	1.09
8	Proposed	12	35	21	68	1.00

The gate level evaluations of competitive designs are generally accepted only as rough estimates and actual realizations may lead to reordering of performance figures of the designs under consideration. This will be duly examined below.

4.2. Synthesis-based results and comparisons

Comparison of synthesis-based performance measures is best accomplished when all designs are synthesized with the same technology files under the same working conditions. Therefore, we have used typical TSMC 130nm technology by Synopsis Design Compiler to synthesize all designs, except that of [27], for which sufficient details are not available. However, since latency of the work of [27] is compared therein with that of [2], as 2.51 ns versus 2.65 ns, we have scaled our synthesis results for the latter based on the improvement ratio $\frac{2.51}{2.65} = 0.95$ to get at reliable measures for performance of the former. These designs have been verified for correctness via sufficiently large random test vectors as well as manually generated vectors for corner cases. Figs. 12 and 13 show the results for area consumption and power dissipation with time constraints from the minimum that could be met, by the proposed design (i.e., indeed the least among all), up to 10 ns by 0.2 ns steps. Regarding the work of [27], the minimum time constraint that it could meet, would be 5% less than that of [2] (i.e., 5ns), which can be obtained as $5 \times 0.95 = 4.75$ ns, while that of the proposed design is 4.4 ns.

Inspecting Fig. 12 shows that the proposed design can perform with latency as low as 4.4 ns, while the next lower time constraint is 4.8 ns, which is due to [4], with almost the same area consumption (actually 1.5% more than the proposed design) on the same 4.8 ns time point. Therefore, the synthesis based 8.3% latency improvement of the proposed design confirms the 8.1% less latency that is experienced based on analytical evaluations (i.e., 68 ΔG versus 74 ΔG). At the 4.8 ns point (in Fig. 13), power dissipation of [4] is 10% more than that of the proposed design. The resulted improvements, primarily seems to be due to the saved XOR gates in the proposed PPG architecture. Moreover, the reduced depth of 16 for the partial product matrix and the used redundant adders for PPR have considerably contributed in area and power savings.

V. Conclusion

We propose a parallel 16×16 radix-10 BCD multiplier, where 17 partial products are generated with $[-6, 6]$ sign-magnitude signed digit (SMSD) representation. Some innovations of this work and use of previous techniques, as listed below, has led to marginal 1.5% less area consumption, and 10% less power dissipation, on 4.8 ns latency, with respect to the fastest previous work due to [4]. The least possible delay for the latter is 4.8 ns, while the proposed design leads the synthesis tool to meet 4.4 ns time constraint (i.e., 9% faster). In other words, the advantage is that the proposed design can operate in 9% higher frequency and dissipate up to 13% less power, with no claim in area improvement.

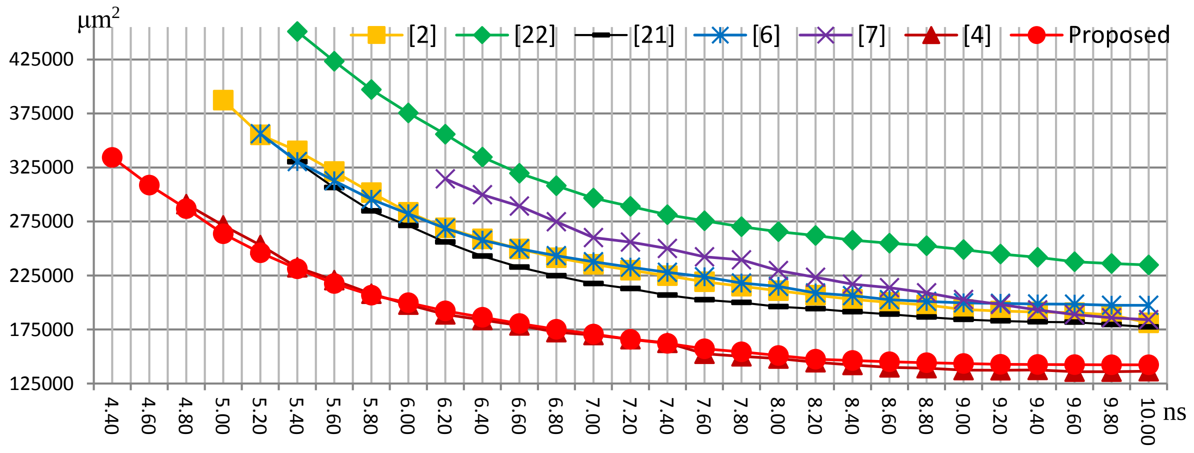


Fig. 12. Comparison of area consumption.

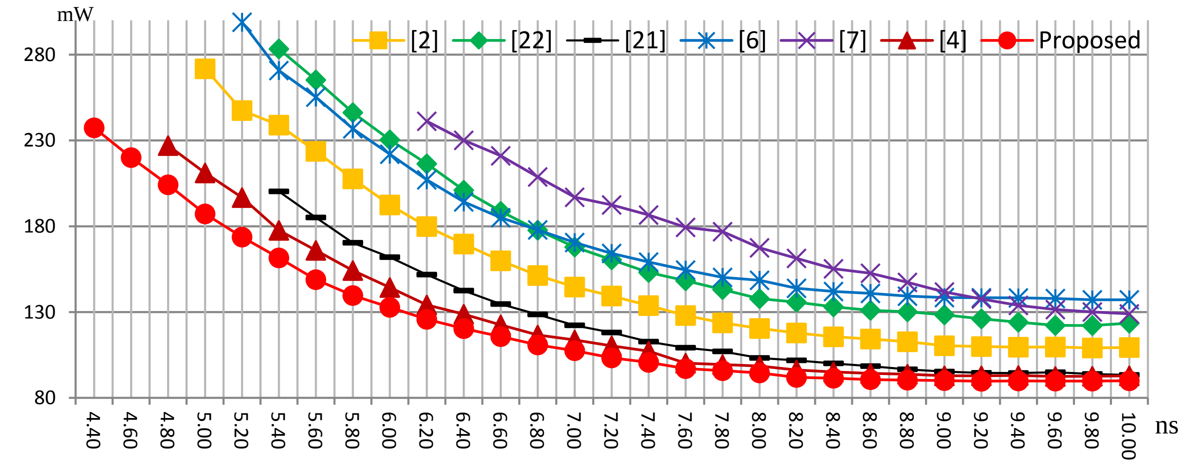


Fig. 13. Comparison of power dissipation.

- ✓ **Sign magnitude signed digit representation**: The exclusively employed SMSD representation of partial products saves more than 850 XOR gates ($\approx 75\%$) in comparison to other 16×16 decimal multipliers with dynamic negation of partial products.
- ✓ **On the fly depth reduction**: Two SMSD digits of the sole deepest column of partial product matrix are reduced to one, in parallel with PPG that leads the VLSI-regular 2:1 PPR to start with 16 partial products.
- ✓ **4-in-1 SMSD-to-TCSD adder**: This is the most novel contribution of the present work. The reason is that sign-magnitude addition conceptually entails separate consideration of four sign combinations. To avoid the corresponding inefficiency, the first-level reduction is undertaken via eight especial SMSD adders. However, enforcing the SMSD signs via polarity of magnitudes has led to a unified 4-in-1 adder logic, which is no more complex than a simple TCSD adder.
- ✓ **Early initiation of redundant-to-BCD conversion**: To take advantage of early signal arrivals, conversion of the four least significant digits to BCD starts in the middle of PPR. A parallel prefix compound Kogge-Stone adder produces BCD sum and sum -1 for the nine most significant digits.
- ✓ **Parallel prefix carry select addition**: A special parallel prefix decimal carry select adder adds up the middle TCSD digits and produces BCD sum digits and a borrow that selects one of the two BCD sums of the most significant part.

Future research in the line of this work can include similar sequential multiplication scheme, and use of this multiplier in decimal floating-point units. Also, manufacturing perspective of parallel decimal multipliers may be considered as strengthened due to smaller size and lower power of such improved designs.

APPENDIX

The logical equations for the bits of multiples $\{1, 2, 4, 5\}$ of the multiplicand X are presented in Eqn. sets A1 to A4, respectively. These equations are derived in the similar way to that $3X$, which was described in Section 3.2.

$$\begin{aligned}
s_u &= b_2 \vee b_3\left(\overline{b}_0 \vee \overline{a}_3\overline{a}_2\right)\\
u_2 &= \overline{b}_2 b_1 b_0 (a_3 \vee a_2) \vee b_2\left(\overline{b}_1 \vee \overline{a}_3\overline{a}_2\overline{b}_0\right)\\
u_1 &= \overline{a}_3\overline{a}_2\left(b_3\overline{b}_0 \vee b_1 b_0 \vee b_2\overline{b}_1\overline{b}_0\right) \vee\\
&\quad (a_3 \vee a_2)\left(b_2 b_1 \vee \overline{b}_3\overline{b}_2\overline{b}_1 b_0\right) \vee \overline{b}_2 b_1 \overline{b}_0\\
u_0 &= \overline{b}_0(a_3 \vee a_2) \vee \overline{a}_3\overline{a}_2 b_0
\end{aligned} \tag{A1}$$

$$\begin{aligned}
s_d &= \overline{a}_3\left(b_3 \vee b_2\overline{b}_1\overline{b}_0\right) \vee \overline{b}_2 b_1 \vee b_3\overline{b}_0\\
d_2 &= \overline{a}_3\overline{a}_2\overline{a}_1\left(\overline{b}_2 b_1 \vee b_3\overline{b}_0\right) \vee b_1\overline{b}_0\left(\overline{b}_2 \vee a_3\right) \vee\\
&\quad b_0\left(b_2 b_1 \vee a_3\overline{b}_3\overline{b}_2\overline{b}_1\right)\\
d_1 &= \overline{a}_3\overline{a}_2\overline{a}_1\left(b_1\overline{b}_0 \vee b_2\overline{b}_0 \vee \overline{b}_2\overline{b}_1 b_0\right) \vee\\
&\quad a_3\left(b_1 b_0 \vee b_2 b_0 \vee \overline{b}_2\overline{b}_1\overline{b}_0\right) \vee\\
&\quad (a_2 \vee a_1)\left(b_3\overline{b}_0 \vee \overline{b}_3\overline{b}_2 b_0\right) \vee a_3 b_2 b_1 \overline{b}_0\\
d_0 &= a_2 \vee a_1
\end{aligned} \tag{A2}$$

$$\begin{aligned}
s_q &= \overline{a}_3\overline{a}_2 b_1\left(b_2 \vee \overline{b}_0\right) \vee b_2\overline{b}_0(\overline{a}_3 \vee \overline{a}_0 \vee b_1) \vee\\
&\quad b_0\left(b_3(\overline{a}_3 \vee \overline{a}_0) \vee \overline{b}_3\overline{b}_2\overline{b}_1\right)\\
q_2 &= \overline{a}_3\overline{a}_2\overline{a}_1\overline{a}_0\left(b_2\overline{b}_0 \vee \overline{b}_2\overline{b}_1 b_0\right) \vee a_3 a_0 \overline{b}_1\left(\overline{b}_2\overline{b}_0 \vee b_2 b_0\right) \vee\\
&\quad \overline{a}_3\overline{a}_2\overline{a}_1\left(b_2 b_1 \overline{b}_0 \vee \overline{b}_3\overline{b}_2\overline{b}_1 b_0\right) \vee\\
&\quad (a_3 \vee a_2)\left(b_3\overline{b}_0 \vee \overline{b}_2 b_1 b_0\right)\\
q_1 &= \overline{b}_3\overline{b}_1 b_0\left(a_3\overline{a}_0 \vee a_3\overline{b}_2 \vee a_2 a_1\right) \vee\\
&\quad b_2\overline{b}_1\overline{b}_0(\overline{a}_2 a_1 \vee \overline{a}_3\overline{a}_1 a_0) \vee\\
&\quad \overline{a}_3\overline{a}_2\left(\overline{a}_1\overline{a}_0\left(b_1 \vee \overline{b}_3\overline{b}_2 b_0\right) \vee b_3\left(a_0 \vee \overline{b}_0\right) \vee \overline{b}_2 b_1 b_0\right) \vee\\
&\quad a_2\left(\overline{a}_1\left(b_2\overline{b}_1 \vee b_3 b_0\right) \vee \overline{b}_0\left(\overline{b}_3\overline{b}_2\overline{b}_1 \vee a_1 b_2 b_1\right)\right) \vee\\
&\quad \overline{a}_2 b_3\left(a_1 \vee a_0\overline{b}_0\right) \vee a_3\left(b_1\left(a_0 \vee b_2\overline{b}_0\right) \vee \overline{a}_0\overline{b}_3\overline{b}_2\overline{b}_1\right)\\
q_0 &= a_1 \vee a_3\overline{a}_0 \vee \overline{a}_3\overline{a}_2 a_0
\end{aligned} \tag{A3}$$

$$\begin{aligned}
s_p &= b_0(\overline{a}_3 \vee \overline{a}_1)\\
p_2 &= \overline{a}_3\overline{a}_2 b_0(\overline{a}_1 \vee \overline{a}_0) \vee \overline{b}_0(a_3 \vee a_2 a_1 a_0)\\
p_1 &= a_2 \oplus a_1 a_0,\\
p_0 &= a_1 \oplus a_0 \oplus b_0
\end{aligned} \tag{A4}$$

ACKNOWLEDGMENT

The authors wish to sincerely appreciate the efforts put by the anonymous reviewers to pinpoint all the shortcoming of the original manuscript. This research was supported in part by IPM under Grant CS1395-2-03, and in part by Shahid Beheshti University.

REFERENCES

[1] Cowlishaw, M. F., "Decimal Floating-Point: Algorism for Computers," *In Proc. 16th IEEE Symposium on Computer Arithmetic*, pp. 104-111, Jun. 2003.

[2] Lang, T. and A. Nannarelli, "A Radix-10 Combinational Multiplier," *In Proc. 40thAsilomar Conference on Signals, Systems, and Computers*, Nov. 2006.

[3] Kenney, R. D., M. J. Schulte and M. A. Erle, "A High-Frequency Decimal Multiplier," *In Proc. IEEE International Conference on Computer Design (ICCD)*, pp. 26-29, Oct. 2004.

[4] Vazquez A., E. Antelo, J. D. Bruguera, "Fast Radix-10 Multiplication Using Redundant BCD Codes," *IEEE Transactions on Computers*, Vol. 63, No. 8, pp. 1902-1914, Apr. 2014.

[5] Gorgin S. and G. Jaberipur, "A Fully Redundant Decimal Arithmetic," *In Proc. 19th IEEE Symposium on Computer Arithmetic*, pp. 145-152, 2009.

[6] Vazquez A., E. Antelo, and P. Montuschi, "Improved Design of High-Performance Parallel Decimal Multipliers," *IEEE Transactions on Computers*, Vol. 59, No. 5, pp. 679-693, May 2010.

[7] Han L. and S. Ko, "High Speed Parallel Decimal Multiplication with Redundant Internal Encodings," *IEEE Transactions on Computers*, 10.1109/TC.2012.35, Jan. 2012.

[8] Jaberipur G. and A. Kaivani, "Binary-Coded Decimal Digit Multipliers," *IET Computers & Digital Techniques*, Vol. 4, pp. 377-381, 2007.

[9] James R. K., T. K. Shahana, K.P. Jacob, and S. Sasi, "Decimal Multiplication Using Compact BCD Multiplier," *In Proc. the International Conference on Electronic Design*, pp. 1-6, 2008.

[10] Erle, M. A., E. M. Schwartz, and M. J. Schulte, "Decimal Multiplication with Efficient Partial Product Generation," *In Proc. 17th IEEE Symposium on Computer Arithmetic*, pp. 21-28, Jun. 2005.

[11] Richards R. K., "Arithmetic Operations in Digital Computers," Van Nostrand, New York, 1955.

[12] IEEE Standards Committee, 754-2008 IEEE Standard for Floating-Point Arithmetic, 2008. DOI: 10.1109/IEEESTD.2008.4610935

[13] Svoboda A., "Decimal Adder with Signed Digit Arithmetic," *IEEE Transactions on Computers*, Vol. C-18, No. 3, pp. 212-215, Mar. 1969.

[14] Hickmann B. J., A. Krioukov, M. J. Schulte, and M. A. Erle, "A Parallel IEEE P754 Decimal Floating-Point Multiplier," *In Proc. IEEE International Conference on Computer Design (ICCD)*, pp. 296-303, 2007.

[15] Raafat R., A.M. Abdel-Majeed, R. Samy, T. ElDeeb, Y. Farouk, M. Elkhouly, and H.A.H Fahmy, "A Decimal Fully Parallel And Pipelined

Floating-Point Multiplier," *In Proc. 42th Asilomar Conference on Signals, Systems, and Computers*, pp. 1800-1804, Oct. 2008.

[16] Erle M. A., B. J. Hickmann, and M. A. Schulte, "Decimal Floating-Point Multiplication," *IEEE Transactions on Computers*, Vol. 58, No. 7, pp. 902-916, Jul. 2009.

[17] Tsen C., S. Gonzalez-Navarro, M. Schulte, B. Hickmann, and K. Compton, "A Combined Decimal and Binary Floating-Point Multiplier," *In Proc. Application-specific Systems, Architectures and Processors (ASAP)*, pp. 8-15, Jul. 2009.

[18] Minchola C. and G. Sutter, "A FPGA IEEE-754-2008 Decimal64 Floating-Point Multiplier," *In Proc. the International Conference on Reconfigurable Computing and FPGAs, (ReConFig '09)*, pp. 59- 64, Dec 2009.

[19] Vazquez A. and F. Dinechin, "Efficient Implementation of Parallel BCD Multiplication in LUT-6 FPGAs," *In Proc. Field-Programmable Technology (FPT)*, pp. 126-133, Dec. 2010.

[20] Erle M. A., and M. J. Schulte, "Decimal Multiplication via Carry-Save Addition," *In Proc. Application-Specific Systems, Architectures, and Processors (ASAP)*, pp. 348-358, Jun. 2003.

[21] Gorgin S. and G. Jaberipur, "A fully redundant decimal adder and its application in parallel decimal multipliers," *Microelectronics Journal*, vol. 40, No. 10, Oct. 2009.

[22] Jaberipur G. and A. Kaivani, "Improving the Speed of Parallel Decimal Multiplication," *IEEE Transactions on Computers*, Vol. 58, No.11, pp. 1539-1552, Nov. 2009.

[23] Shirazi B., D.Y. Yun, and C.N. Zhang, "RBCD: Redundant Binary Coded Decimal Adder," *IEE Proc. Computer & Digital Techniques (CDT)*, Vol.36, No.2, Mar. 1989.

[24] Jaberipur, G. and B. Parhami, "Efficient Realization of Arithmetic Algorithms with Weighted Collections of Posibits and Negabits," *IET Computers & Digital Techniques*, Vol. 6, No.5, pp. 259-268, Sep.2012.

[25] Oklobdzija V. G. and D. Villeger, "Improving Multiplier Design By Using Improved Column Compression Tree And Optimized Final Adder In CMOS Technology", *IEEE Transactions on VLSI Systems*, Vol. 3, No. 2, pp. 292-301, Jun. 1995.

[26] Peter M. Kogge and Harold S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," IEEE Transactions on Computers, C-22, pp. 783-791, 1973.

[27] Dadda L. and A. Nannarelli, "A Variant of a Radix-10 Combinational Multiplier," *In Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '08)*, pp. 3370-3373, May 2008.

[28] Gorgin S. and G. Jaberipur, "Comment on "High Speed Parallel Decimal Multiplication with Redundant Internal Encodings"," *IEEE Transactions on Computers*, 22 Aug. 2013. http://doi.ieeecomputersociety.org/10.1109/TC.2013.160.

[29] Schmookler M., and A. Weinberger, "High Speed Decimal Addition," *IEEE Transactions on Computers*, Vol. C-20, No. 8, pp. 862-866, Aug. 1971.

Saeid Gorgin received BS and MS degrees in computer engineering from the South branch, and the Science and Research branch, of Azad University of Tehran in 2001 and 2004, respectively. He received his Ph.D. degrees in Computer System Architecture from Shahid Beheshti University, Tehran, Iran in 2010. He is currently an assistant professor of Computer Engineering in Department of Electrical Engineering and Information Technology of Iranian Research Organization for Science and Technology (IROST), Tehran, Iran. His research interests include computer arithmetic and VLSI design.

Ghassem Jaberipur, is an Associate Professor of Computer Engineering in the Department of Computer Science and Engineering of Shahid Beheshti University, Tehran, Iran. He received his BS in electrical engineering and PhD in computer engineering from Sharif University of Technology in 1974 and 2004, respectively, MS in engineering from UCLA in 1976, and MS in computer science from University of Wisconsin, Madison, in 1979. His main research interest is in computer arithmetic. Dr. Jaberipur is also affiliated with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), in Tehran, Iran.