Hindawi
International Journal of Reconfigurable Computing
Volume 2017, Article ID 2410408, 12 pages
https://doi.org/10.1155/2017/2410408

Research Article

Efficient Realization of BCD Multipliers Using FPGAs

Shuli Gao,[1] Dhamin Al-Khalili,[1] J. M. Pierre Langlois,[2] and Noureddine Chabini[1]

[1] *Department of Electrical and Computer Engineering, Royal Military College of Canada, Kingston, ON, Canada*
[2] *Department of Computer Engineering, École Polytechnique de Montréal, Montréal, QC, Canada*

Correspondence should be addressed to Dhamin Al-Khalili; alkhalili-d@rmc.ca

Received 22 October 2016; Revised 2 February 2017; Accepted 9 February 2017; Published 6 March 2017

Academic Editor: Seda Ogrenci-Memik

Copyright © 2017 Shuli Gao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In this paper, a novel BCD multiplier approach is proposed. The main highlight of the proposed architecture is the generation of the partial products and parallel binary operations based on 2-digit columns. 1×1-digit multipliers used for the partial product generation are implemented directly by 4-bit binary multipliers without any code conversion. The binary results of the 1×1-digit multiplications are organized according to their two-digit positions to generate the 2-digit column-based partial products. A binary-decimal compressor structure is developed and used for partial product reduction. These reduced partial products are added in optimized 6-LUT BCD adders. The parallel binary operations and the improved BCD addition result in improved performance and reduced resource usage. The proposed approach was implemented on Xilinx Virtex-5 and Virtex-6 FPGAs with emphasis on the critical path delay reduction. Pipelined BCD multipliers were implemented for 4×4, 8×8, and 16×16-digit multipliers. Our realizations achieve an increase in speed by up to 22% and a reduction of LUT count by up to 14% over previously reported results.

1. Introduction

The traditional approach of using binary number system based operations in a decimal system requires frontend and backend conversion. These conversions can take a significant amount of processing time and consume large area. A more important problem with fractional decimal numbers expressed in a binary format may result in lack of accuracy. This can have major impact in finance and commercial applications. To solve these problems, interest in hardware design of decimal arithmetic is growing. This has led to the incorporation of specifications of decimal arithmetic in the IEEE-754 2008 standard for floating-point arithmetic [1]. The development of decimal operations in hardwired designs with high performance and low resource usage is expected to facilitate the implementation of various applications [2].

Multiplication is a complex operation among decimal computations. To speed up this operation, early decimal multipliers were designed at the gate level targeting ASICs. The authors in [3] proposed an improved iterative decimal multiplier approach to reduce the number of iteration cycles. To avoid a large number of decimal to binary conversions, a two-digit stage was used as the basic block for the iterative Binary Coded Decimal (BCD) multiplier. To further speed up the multiplication, parallel decimal multipliers were proposed. Binary multiplier and binary to BCD conversion were utilized to implement 1×1-digit multipliers, and different binary compressors were employed for the result of the multiplier [4–6]. To avoid the binary to decimal conversion, recoding methods were used to generate the partial products of the BCD multiplier [7, 8]. A Radix10 combinational multiplier was introduced in [7] and Radix4 and Radix5 recoding methods were presented in [8]. In [9], Radix5 recoding was combined with BCD code converters using BCD4221 and BCD5211 codes to simplify the partial product generation and reduction. In the recent two years, some ASIC-based designs for the realization of decimal multiplication were proposed in [10–14]. The recoding methods and BCD code conversions were used in these designs for efficient implementation in ASIC.

Although there are a number of approaches to implement decimal multipliers in ASICs, utilizing the same methods in FPGA devices is not necessarily efficient. With recent advancements in FPGA technology, enhanced architectures, and availability of various hardware resources, the FPGA

platform is recognized as a viable alternative to ASICs in many cases. To make efficient use of FPGA resources in the implementation of decimal multiplication, new algorithms and approaches have been developed. The authors in [15] implemented decimal multipliers using embedded binary multiplier blocks in FPGAs. The binary-BCD conversion was implemented using base-1000 as an intermediate base, and the result was converted to BCD using a shift-add-3 algorithm. In [16], the authors presented a double-digit decimal multiplier technique that performs 2-digit multiplications simultaneously in one clock cycle; then the overall multiplication was performed serially. In [17, 18], a 1×1-digit multiplier was designed directly with BCD inputs/outputs and implemented using 6-input or 4-input LUTs. To sum the results of 1×1-digit multipliers, a fast carry-chain decimal adder was also proposed in [18]. These decimal-operation-based approaches avoided the conversions but also impacted the speed. Vázquez and De Dinechin implemented a BCD multiplier using a recoding technique [19]. Signed-Digit (SD) Radix5 was employed to recode one of the input operands of the multiplier for the generation of the partial products. 6-input LUTs and fast carry chains in Xilinx FPGAs were used to generate the building blocks and the decimal adders. To increase the performance, the authors in [20] implemented a parallel decimal multiplier based on Karatsuba-Ofman algorithm. The building blocks used in Karatsuba-Ofman algorithm were deigned based on the approach proposed in [19]. Another SD-based decimal multiplier approach was proposed in [21]. The recoding was based on SD Radix10. BCD4221, 5211, and 5421 converters were used for the partial product generation. BCD4221-based compressors and adders were utilized in this approach. Although the BCD4221-based operations are similar to binary operation, the recoding and the different code conversions still lead to delay and resource cost.

In this paper, we propose a new parallel binary-operation-based decimal multiplier approach. Binary operations are performed for the 1×1-digit multiplication and the partial product reduction based on the columns with two digits in each column. The operations for all columns are processed in parallel. After the column-based binary operations, binary to decimal conversions are required but the bit sizes of the operands to be converted are limited based on the columns. In this paper, an improved 6-LUT-based BCD adder and a 2-digit column-based binary-decimal compressor are also presented. Our proposed approach was implemented in Xilinx Virtex-5 and Virtex-6 FPGAs. The results are compared with Radix-recoding-based approaches using a BCD4221 coding scheme. The proposed approach achieves improved FPGA performance in part because of the parallel binary operations and small size conversions.

The organization of this paper is as follows. Section 2 presents optimized building blocks required by the BCD multiplication. The proposed multiplier architecture and the schemes of the partial product generation and reduction are presented in Section 3. The implementation results of $n \times n$-digit BCD multipliers are depicted in Section 4. Conclusions are given in Section 5.

2. Proposed Building Blocks for the Realization of BCD Multiplication

In this section, proposed schemes for an improved 6-input LUTs-based BCD adder and a mixed binary-decimal compressor are presented. These schemes will be utilized as the basic building blocks to construct our proposed BCD multipliers presented in Section 3.

2.1. 6-Input LUTs-Based 1-Digit BCD Adder. The 6-input LUTs-based 1-digit BCD adder is based on the use of 6-input LUTs and MUX-XOR networks in FPGAs. It is an improved version of the architecture presented in [19].

Assume that the input operands of the adder are $A = [a_3\ a_2\ a_1\ a_0]$ and $B = [b_3\ b_2\ b_1\ b_0]$ in BCD8421 format. The input operands are decomposed as

$$\begin{aligned} A &= [a_3\ a_2\ a_1] \times 2 + a_0 = A_1 \times 2 + a_0, \\ B &= [b_3\ b_2\ b_1] \times 2 + b_0 = B_1 \times 2 + b_0. \end{aligned} \tag{1}$$

Then, the addition is presented as

$$\begin{aligned} A + B + C_{\text{in}} &= (A_1 \times 2 + a_0) + (B_1 \times 2 + b_0) + C_{\text{in}} \\ &= (A_1 + B_1) \times 2 + (a_0 + b_0 + C_{\text{in}}) \\ &= [F_4\ F_3\ F_2\ F_1] \times 2 + [C_0 \times 2 + S_0]. \end{aligned} \tag{2}$$

In (2), A_1 or B_1 has the binary set $\{000, 001, 010, 011, 100\}$, and the full adder $[a_0 + b_0 + C_{\text{in}}]$ has two outputs, the carry C_0 and the sum S_0. The function $F = [F_4\ F_3\ F_2\ F_1]$ is a three-bit adder with the add-3 correction merged, which can be expressed as

$$F = [F_4\ F_3\ F_2\ F_1] = \begin{cases} A_1 + B_1 & \text{if } A_1 + B_1 < 5, \\ A_1 + B_1 + 3 & \text{if } A_1 + B_1 \geqslant 5. \end{cases} \tag{3}$$

In (3), the F cannot be $[0101]_2$, $[0110]_2$, or $[0111]_2$ because of the +3 correction. Also, since the maximal value of A_1 and B_1 is $[100]_2$, the maximal value of F is $[1011]_2$. The function F has 6 inputs; therefore, it can be efficiently mapped in a single level of 6-input LUTs.

To calculate the final result in BCD format, the carry C_0 of the full adder must be added to F. As a special case, an add-3 correction must be considered if $F = 4$ and $C_0 = 1$ to achieve a correct final result. Table 1 is the truth table for the final correction.

Therefore, the proposed scheme requires the following steps:

(i) Decompose the addition as two adders: one is a full adder for adding the two least significant bits of the input operands with the incoming carry, and another is a 3-bit adder with add-3 correction merged for the remaining bits. This function decomposition is presented in (2).

(ii) Implement the full adder and the 3-bit adder merged with an add-3 correction as presented in (3).

Table 1: Final correction for the BCD adder.

$F = F_4\ F_3\ F_2\ F_1$	$C_0 = 0$	$C_0 = 1$	Comments
	$C_{out}\ S_3\ S_2\ S_1$	$C_{out}\ S_3\ S_2\ S_1$	
0 0 0 0	0 0 0 0	0 0 0 1	"+3" is not required
0 0 0 1	0 0 0 1	0 0 1 0	"+3" is not required
0 0 1 0	0 0 1 0	0 0 1 1	"+3" is not required
0 0 1 1	0 0 1 1	0 1 0 0	"+3" is not required
0 1 0 0	0 1 0 0	1 0 0 0	*at $C_0 = 1$, "+3" is required*
0 1 0 1	x x x x	x x x x	
0 1 1 0	x x x x	x x x x	
0 1 1 1	x x x x	x x x x	
1 0 0 0	1 0 0 0	1 0 0 1	"+3" has been performed
1 0 0 1	1 0 0 1	1 0 1 0	"+3" has been performed
1 0 1 0	1 0 1 0	1 0 1 1	"+3" has been performed
1 0 1 1	1 0 1 1	1 1 0 0	"+3" has been performed
1 1 0 0	x x x x	x x x x	
1 1 0 1	x x x x	x x x x	
1 1 1 0	x x x x	x x x x	
1 1 1 1	x x x x	x x x x	

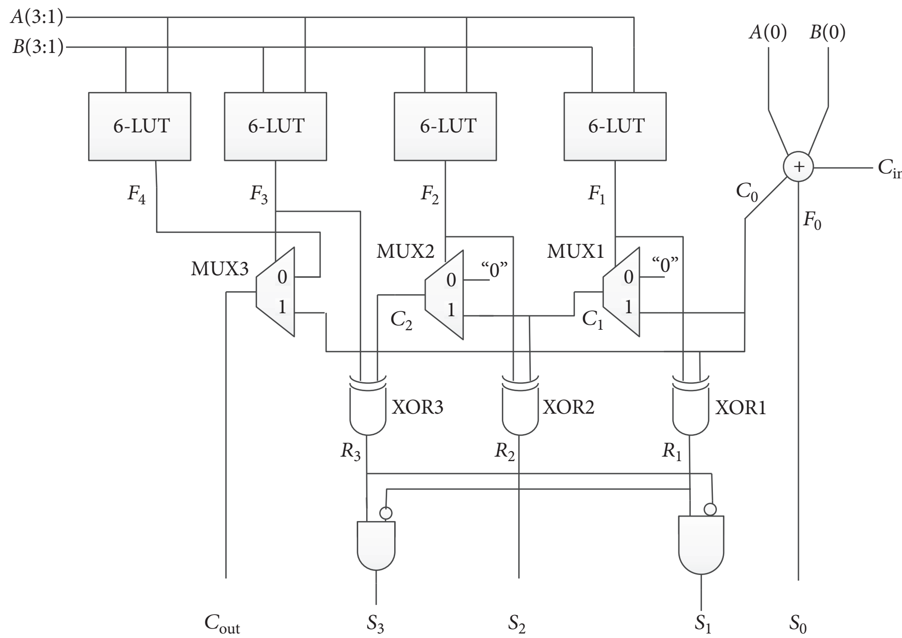


Figure 1: Improved 1-digit BCD adder using 6-LUTs and MUX-XOR network in FPGA.

(iii) Add the carry of the full adder with the output of the 3-bit adder using MUX-XOR networks. The multiplexers generate the propagated carries and the XOR gates output the sum bits.

(iv) Perform a final correction for the case of the carry of the full adder equal to "1" and the sum of the 3-bit adder equal to "4" to obtain the final result.

Figure 1 shows the architecture of this approach.

In this design, if the carry of the full adder, C_0, is "0"; there is no change to the result of the 3-bit adder and no carry is propagated. The output of the BCD adder is the same as that of the 3-bit adder, which is $[C_{out}\ S_3\ S_2\ S_1] = [F_4\ F_3\ F_2\ F_1]$. However, if C_0 is "1," the carry must be added to the result of the 3-bit adder. First, XOR_1 and MUX_1 add C_0 to F_1 and generate the sum $R_1 = (F_1 \text{ XOR } C_0)$ and the carry $C_1 = (F_1 \text{ AND } C_0)$. If $C_0 = 1$ and $F_1 = 0$, the sum R_1 is equal to "1" and no carry ($C_1 = 0$) is propagated. However, if $C_0 = 1$ and $F_1 = 1$, the sum R_1 is equal to "0," and the carry is propagated to C_1. The same procedure applies to XOR_2 and MUX_2. For MUX_3, it produces the output carry, C_{out}. Based on the truth table listed in Table 1, the output carry C_{out} is the same as F_4 when $F_3 = 0$ and the same as C_0 when $F_3 = 1$, which is realized by MUX_3. In this case, propagating C_0 from the output of the

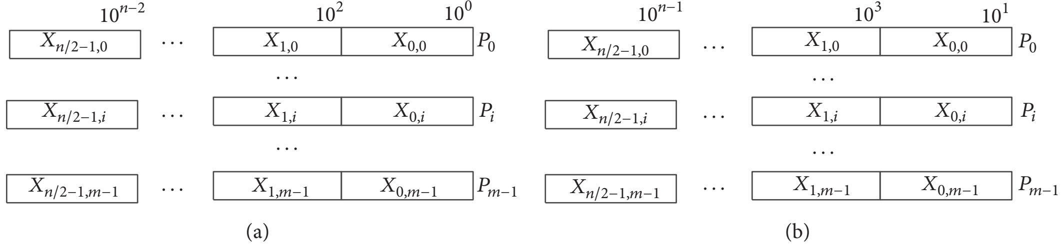


Figure 2: Two-group operands with the mixed binary-decimal format.

Table 2: Comparison of the implementation results for the BCD adders.

Improved 6-LUT		Reference [19]	
Delay (ns)	LUTs	Delay (ns)	LUTs
1.372	10	1.397	10

full adder directly to the input of MUX_3 reduces this critical path delay. This has a significant performance impact on large size BCD ripple adders required by BCD multipliers.

To achieve a correct final result, a final correction in the cases of $C_0 = 1$ and $F = 4$ must be performed to the sum. Since, before the final correction, the sum of the adder is equal to

$$\begin{aligned}([F_4\ F_3\ F_2\ F_1] + C_0) \times 2 + F_0 &= (0100 + 1) \times 2 + F_0 \\ &= (101\ F_0)_2 = (R_3\ R_2\ R_1\ F_0)_2,\end{aligned} \tag{4}$$

therefore under the condition of $C_0 = 1$ and $F = 4$, the final add-3 correction is performed to ($R_3\ R_2\ R_1$), and the final result is equal to

$$[C_{\text{out}}\ S_3\ S_2\ S_1\ F_0] = (1\ 000\ F_0)_{\text{BCD}}. \tag{5}$$

In this case, the outputs, S_3 and S_1, have to be forced to “0.” Otherwise, S_3 and S_1 are the same as R_3 and R_1, respectively. Thus, the final correction performed to S_3 and S_1 is equal to

$$\begin{aligned}S_3 &= R_3 \cdot \overline{R_3 \cdot R_1} = R_3 \cdot \overline{R_1}, \\ S_1 &= R_1 \cdot \overline{R_3 \cdot R_1} = R_3 \cdot \overline{R_1}.\end{aligned} \tag{6}$$

The proposed 1-digit BCD adder was coded in VHDL and implemented in a Virtex-6 6vlx75tff784 Xilinx FPGA with a −3 speed grade using ISE13.1 [23]. The results are compared with the carry-ripple BCD adder approach proposed in [19] using the same FPGA. The delays were extracted from Postplacement-and-Routing Static Timing Report and the LUTs usage was obtained from Place-and-Routing Report. Table 2 lists the implementation results.

Table 2 shows that the improved 6-LUT-based BCD adder approach achieves better performance compared with the reference BCD adder. Although the improvement in delay is approximately 2%, for large size adders the cumulative effect can be significant.

2.2. Binary-Decimal Compression. The binary-decimal (BD) compression performs 2-digit column-based binary operations and binary to decimal conversions. The input operands of the BD compression are the results of 1×1-digit BCD multipliers presented in binary format, and the output of the BD compression is in BCD format. Since a 1×1-digit BCD multiplier results in a 2-digit decimal number, the binary inputs are based on 2-digit decimal positions. The input operands of the BD compression are

$$P_i = \sum_{k=0}^{n/2-1} X_{k,i} \times 10^{2k} \quad \text{for } i = 0, 1, 2, \ldots, m-1 \tag{7}$$

$$\text{or } P_i = \sum_{k=0}^{n/2-1} X_{k,i} \times 10^{2k+1} \quad \text{for } i = 0, 1, 2, \ldots, m-1, \tag{8}$$

where m is the number of operands to be compressed, and n is the number of digits in each operand. The variable $X_{k,i}$ is expressed in a binary format but placed in a 2-digit decimal position. Since $X_{k,i}$ is the result of a 1×1-digit BCD multiplier, it has 7 binary bits. Figure 2 illustrates these two-group operands, where (a) and (b) correspond to (7) and (8), respectively. The difference between Figures 2(a) and 2(b) is the decimal positions of the columns.

The binary-decimal compression performs the following steps:

(i) Aligning the input operands based on 2-digit decimal position. All operands in the same column should have the same 2-digit decimal position

(ii) Compressing all operands in each of the columns using binary compressors

(iii) Adding the compressed binary operands in each column using binary adders

(iv) For each column, converting the binary sum to decimal with two digits as the sum and other digits as the carry

(v) Saving the decimal sums and carries in carry-save format for all columns based on their decimal positions

As an example, Figure 3 illustrates the BD compression with m input operands for the case presented in Figure 2(a). This procedure can also be used for the case in Figure 2(b).

In this case, the BD compression first compresses the m binary operands to one binary sum using binary compressors

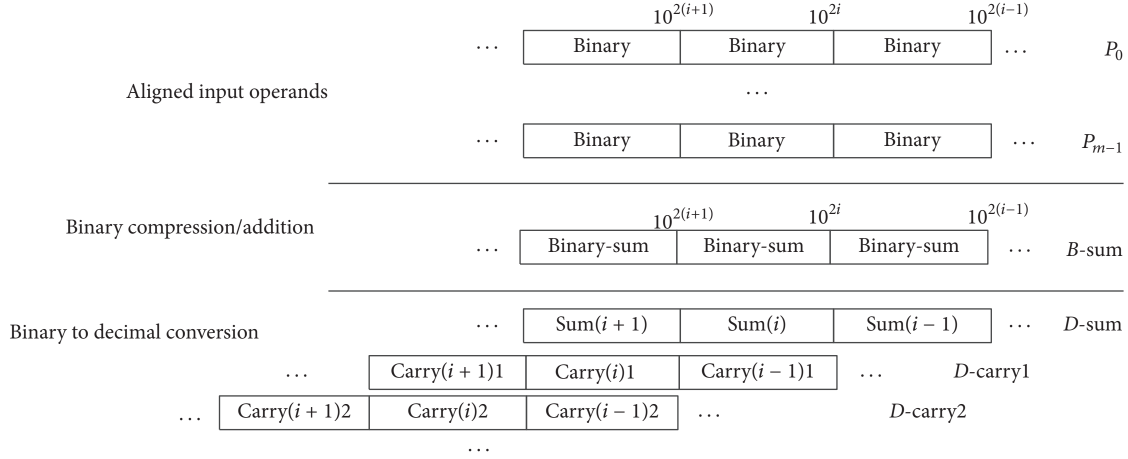


Figure 3: Binary-decimal compression.

and binary adders. In this step, the binary compressors reduce m binary operands to $k = (\lfloor \log_2 m \rfloor + 1)$ operands; then the binary adders add these k operands to produce a binary sum.

Then, the binary sum is converted to a decimal number. The decimal number has a two-digit decimal sum, D-sum(i), and the decimal carries, D-carry$(i)_t$ (for $t = 1, 2, \ldots$). Each of the decimal sums or decimal carries takes two-digit position. The D-sum(i) is located at the 10^{2i} column and the carries are located at the columns of 10^{2i+1}, 10^{2i+2}, and so on. Then, the decimal sum and carries for each column are saved as carry-save format based on their digit positions. Therefore, only $(t + 1)$ decimal operands are generated after the BD compression. The value of t is dependent on the value of m. If m is between 2 and 123, the maximal decimal result in each column is $81 \times 123 = 9963$, for which D-sum = 63 and D-carry = 99. In this case, only t (=1) decimal carry is generated. Thus, 123 such binary operands can be compressed to two decimal operands, one for the D-sum and the other for the D-carry. This arrangement results in a fast way to reduce the number of partial products for a BCD multiplier.

3. Proposed BCD Multiplier Approach

In this section, we present a binary-decimal compression (BDC) based BCD multiplier. The proposed approach consists of 1 × 1-digit binary multiplication, partial product generation, binary-decimal compression, and decimal addition. Figure 4 shows a block diagram which captures all the steps for this approach.

3.1. 1 × 1-Digit Binary Multipliers. The 1 × 1-digit binary multiplier receives two 1-digit BCD operands and outputs a binary result. The maximal output is $9 \times 9 = 81 = [1010001]_2$, which is a 7-bit binary number. Since 1-digit 8421BCD number is the same as a 4-bit binary number, a 4 × 4-bit binary multiplier is used to perform the 1 × 1-digit binary multiplier. In our approach, the 4 × 4-bit binary multiplier is simply coded as $X \times Y$, where X and Y are 1-digit 8421BCD numbers.

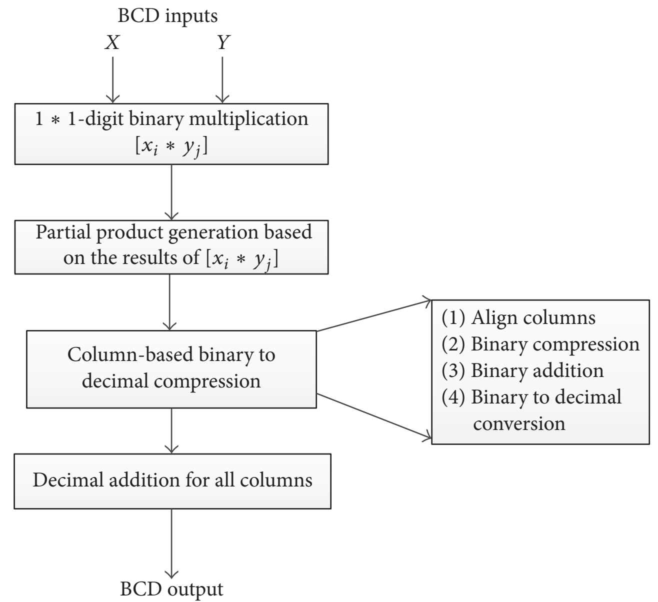


Figure 4: Block diagram of the proposed BDC-based BCD multiplier.

3.2. Partial Product Generation (PPG). The partial product generation is based on 1 × 1-digit binary multipliers. These binary outputs of the 1 × 1-digit binary multipliers are grouped according to their decimal positions. A triangular organization of the partial products is used for the BCD multiplier, which is similar to our previous work proposed in [24] for a binary multiplier. For the BCD multiplication, let us assume that the input operands of the multiplier are X and Y. They are in BCD format and can be expressed as

$$X = \sum_{i=0}^{n-1} X_i \times 10^i, \qquad Y = \sum_{j=0}^{n-1} Y_j \times 10^j. \tag{9}$$

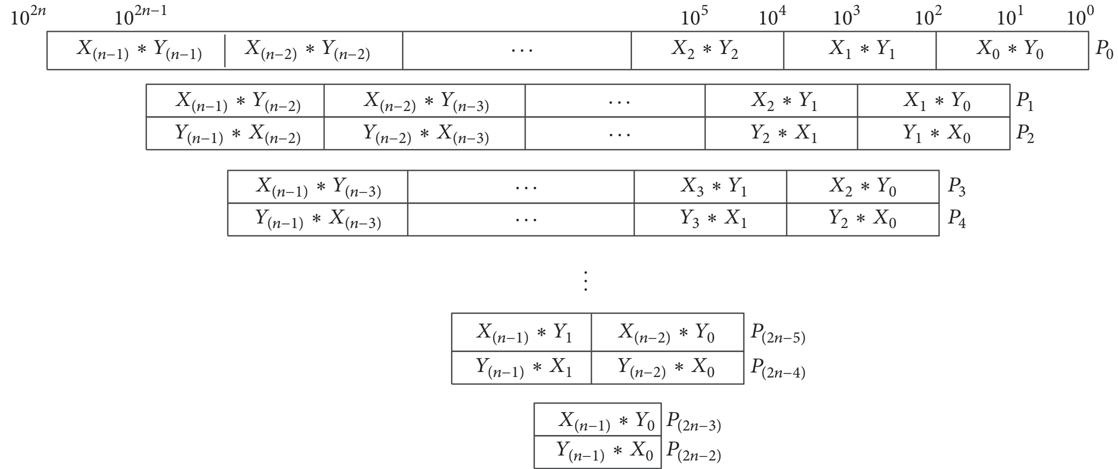


Figure 5: Triangular organization of the partial products of the BCD multiplier.

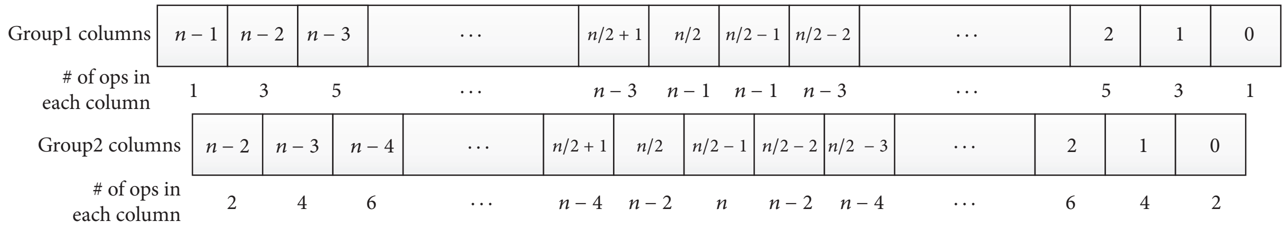


Figure 6: Number of operands in each of the columns.

By multiplying X and Y in (9), the product becomes

$$\begin{aligned} Z = X \times Y &= \left(\sum_{i=0}^{n-1} X_i \times 10^i\right) \times \left(\sum_{j=0}^{n-1} Y_j \times 10^j\right) \\ &= \sum_{i=0}^{n-1} X_i \times Y_i \times 10^{2i} \\ &\quad + \sum_{i=1}^{n-1}\left(\sum_{j=0}^{n-1-i} X_{i+j} \times Y_j \times 10^{(i+2j)}\right) \\ &\quad + \sum_{i=1}^{n-1}\left(\sum_{j=0}^{n-1-i} Y_{i+j} \times X_j \times 10^{(i+2j)}\right), \end{aligned} \tag{10}$$

where $X_i \times Y_i$, $X_{i+j} \times Y_j$, and $Y_{i+j} \times X_j$ are the products from 1×1-digit binary multipliers. These 1×1-digit binary multipliers are organized based on their decimal positions, and the architecture of the BCD multiplier is shown in Figure 5.

Based on the decimal positions of the results of 1×1-digit binary multipliers, these partial products are separated into two groups. The first group is composed of $P_0, P_3, P_4, \ldots, P_{(2n-3)}$ and $P_{(2n-2)}$. The second group is composed of P_1 and $P_2, \ldots, P_{(2n-5)}$ and $P_{(2n-4)}$. The number of operands in each of the columns is shown in Figure 6. The maximal number of operands in the first group is $(n-1)$ that is located at the column $(n/2)$ and column $(n/2-1)$. The maximal number of operands in the second group is n that is located at the column $(n/2-1)$.

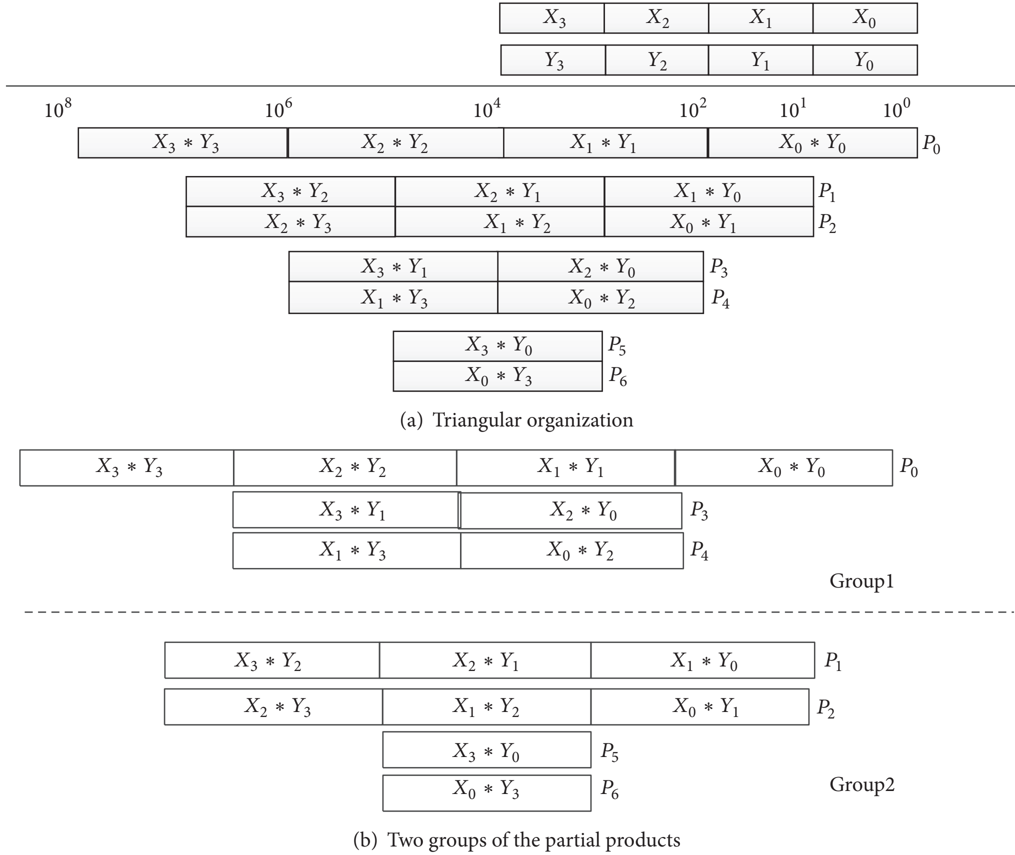
As an example, Figure 7 shows the organization of a 4×4-digit BCD multiplier. In this example, the operands in group 1 are located at the decimal positions 10^{2i} with $i = 0, 1, 2, 3$, and the number of operands in each column is 1, 3, 3, and 1, respectively. The operands in group 2 are located at the decimal positions 10^{2i+1} with $i = 0, 1, 2$, and the number of operands in each column is 2, 4, and 2.

3.3. Partial Product Reduction. Based on the architecture of the BCD multiplier, the partial products are in mixed binary-decimal format. To reduce the number of partial products, two steps are performed: partial product compression and partial product conversion.

Partial Product Compression. The partial product compression performs $(m : 1)$ compression for the binary operands in each column using efficient binary compression and addition methods. The binary compressors first reduce $m = (2^k$ to $2^{k+1} - 1)$ binary operands to $(k + 1)$ binary operands in each column. For example, for $k = 3$ the number of operands to be compressed is $m = (8$ to $15)$. After the compression, 4 binary operands are generated.

Then, these binary operands after the compression are added in binary to obtain a binary sum. Thus, the m operands are compressed to a single one for all columns.

Partial Product Conversion. The partial product conversion converts the binary sum to decimal operands. Double-Dabble (DD) converters [25] can be used in this step. Since the



(a) Triangular organization

(b) Two groups of the partial products

Figure 7: A 4 × 4-digit BCD multiplier.

column-based operations produce limited size binary sums in each column, the conversions introduce only a small delay overhead.

After the binary to decimal conversion, normally only 3 or 4 decimal operands are generated. If there are 12 binary operands or less in one column, the maximal sum is 81 × 12 = 972, which is a 3-digit decimal number. Thus, the decimal sum has two digits and the decimal carry has only one digit. Moreover, the decimal carries in two groups are located at different digital positions. Therefore, the carries can be combined as one decimal operand. Figure 8 illustrates this situation for the 4 × 4-digit BCD multiplier example. Only three decimal operands are generated after the partial product reduction.

However, if there are more than 12 operands in one column, at least four digits are required in this column because 81 × 13 = 1053. Thus, the decimal carry has two digits. In this case, 4 decimal operands will be generated after the partial product reduction. For example, based on the number of operands in each column for a 16 × 16-digit BCD multiplier, as shown in Figure 9(a), the columns at 6, 7, 8, and 9 in the first group create 4-digit decimals for each column, and the columns at 6, 7, and 8 in the second group also generate 4-digit decimals for each column. The decimal operands after the conversion are shown in Figure 9(b), where DS_1 and C_1 are the decimal sum and carry for group 1 and DS_2 and C_2 are the decimal sum and carry for group 2. By combining the decimal carries in two groups, Figure 9(c) shows the decimal operand organization. CC_1 combines the first digit of the carries for all columns, and CC_2 combines the second digit of the carries for the related columns. After partial product reduction, four decimal operands are generated for the 16 × 16-digit BCD multiplier as shown in Figure 9(c).

3.4. Final Decimal Addition (FDA). To obtain the final result of the BCD multiplier, the decimal operands generated after the partial product reduction must be added to decimal adders. BCD ripple adders are used in our approach. These BCD ripple adders are built using our improved 6-LUTs-based BCD adders. Since only 3 or 4 decimal operands need to be added, two-level BCD ripple adders are required. Figure 10 shows the final addition of the BCD multiplication. If there are only three decimal operands to be added, the BCD adder2 in this figure is removed.

3.5. Pipelined Multipliers. Based on the architecture of the BCD multiplier, a 4-stage pipelined BCD multiplier is illustrated in Figure 11.

In this pipelined multiplier, the 1 × 1-digit (4 × 4-bit) binary multiplication and binary compression and addition are combined in the first stage. In this stage, all operations in each column are in binary format. The second stage

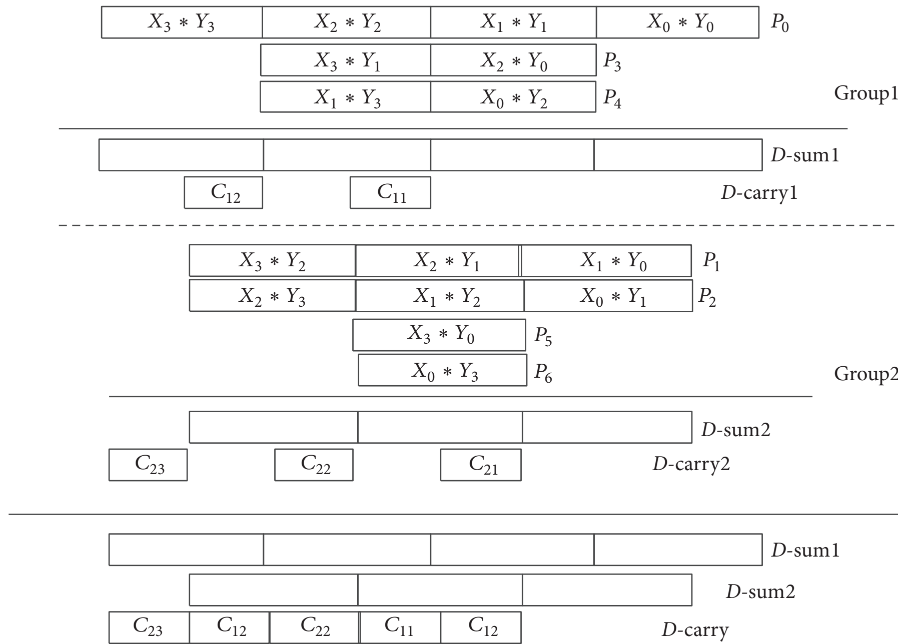


Figure 8: Partial product reduction for a 4 × 4-digit BCD multiplier.

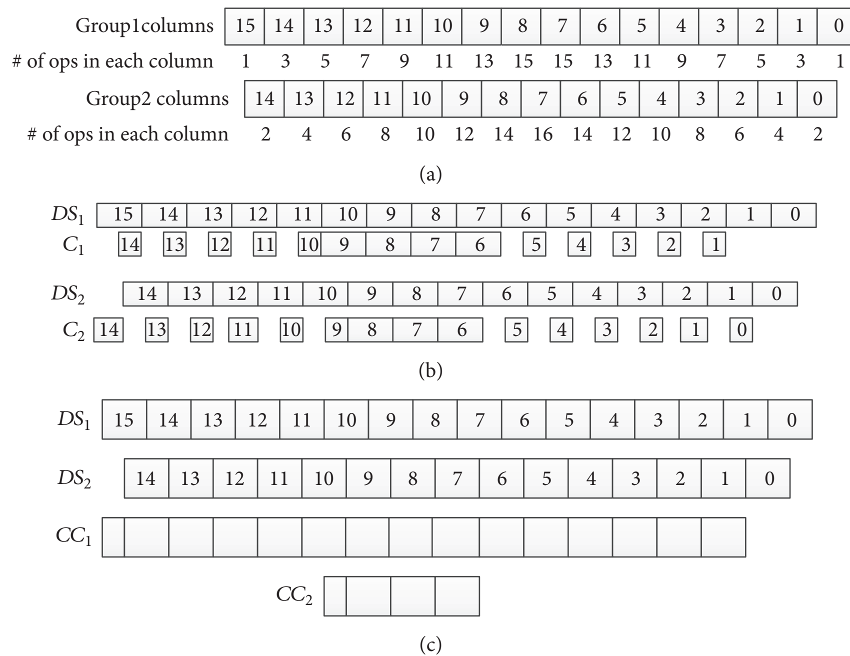


Figure 9: Partial product reduction for a 16 × 16-digit BCD multiplier.

converts the binary numbers to decimal using the Double-Dabble (DD) converter [25]. Since the input operand of the conversion is based on each column, the number of bits in the input operands is limited. Therefore, the delay for the conversions is relatively small. After the binary to decimal conversion, 3 or 4 decimal operands are generated and need to be added. To add these decimal operands, two levels of additions are performed. For a larger size multiplier, more pipeline stages may be required. Figure 12 shows an 8-stage pipeline strategy.

4. Implementation Results

The proposed BCD multiplier approach was implemented in Xilinx Virtex-5 and Virtex-6 FPGAs for 4 × 4, 8 × 8, and 16 × 16-digit pipelined BCD multipliers. The ISE 13.4 tool

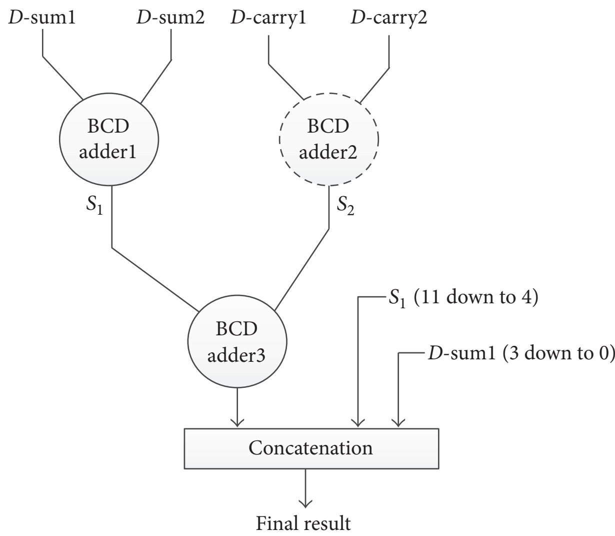


Figure 10: The final addition for a BCD multiplier.

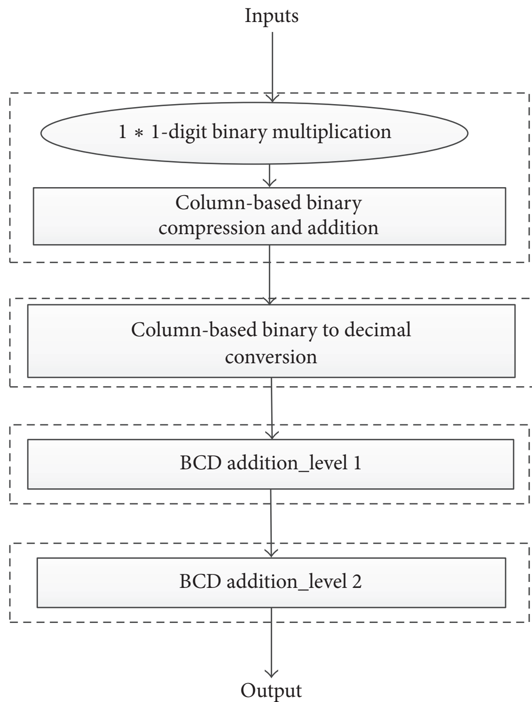


Figure 11: 4-stage pipelined BCD multiplier.

suite [23] was used for the synthesis and implementation. 4×4-bit binary multipliers were used for the partial products generation. The mixed binary-decimal compressors were employed for partial product reduction. The improved 6-LUTs-based BCD adders were connected as ripple adders and used to sum the compressed partial products and generate the final result. Our multipliers were implemented targeting Xilinx xc5vlx330ff1760-2 and xc6vlx760ff1760-2 FPGA devices. The results of the total delay and number of LUTs usage were extracted after the synthesis and implementation and compared with those of the multipliers proposed in [21, 22].

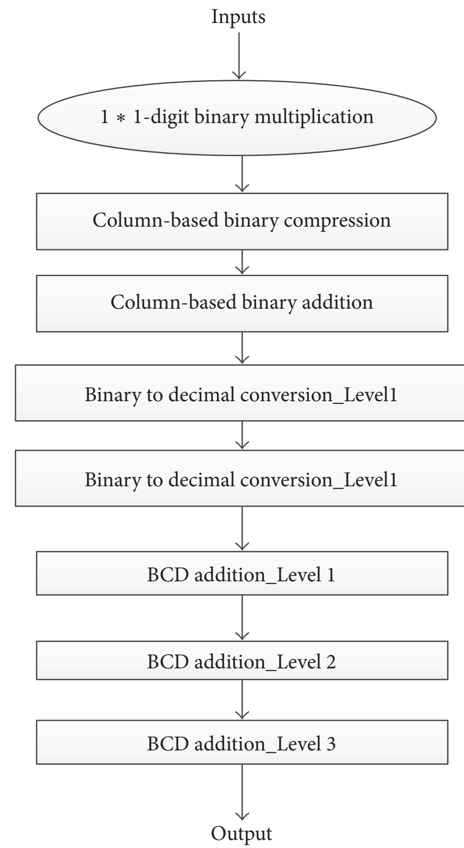


Figure 12: An 8-stage pipeline multiplier.

Figures 13 and 14 illustrate timing information and LUTs utilized for 4×4, 8×8, and 16×16-digit pipelined BCD multipliers based on our proposed approach and on the architecture presented in [21]. The implementation targeted Virtex-5 and Virtex-6 FPGAs, which are the exact same devices used in [21]. The number of pipeline stages was selected based on the best implementation result for each of the multipliers. The total delay, clock cycle time, and LUT usage were depicted in these two figures labeled as (a), (b), and (c), respectively.

Compared with the results presented in [21], our proposed approach achieves improvements in all cases as shown in these figures. On average, the total delay reductions are 22.5% and 14.3% with 14.6% and 16.6% LUT savings when targeting Virtex-5 and Virtex-6 FPGAs, respectively.

The 16×16-digit multiplier with 5, 6, and 7 pipeline stages was implemented targeting Virtex-5 FPGA. The results were compared with the architecture in [22] and presented in Table 3. The total delay of all pipeline stages and the worst-case clock cycle for one pipelined stage were extracted and used for speed comparison.

Compared with the result proposed in [22], our approach achieves faster performance in terms of the total delay and worst-case minimum clock period. On average, the improvement in total delay reduction is 20.2% and in clock cycle reduction is 21.0%, with 8.7% LUTs penalty.

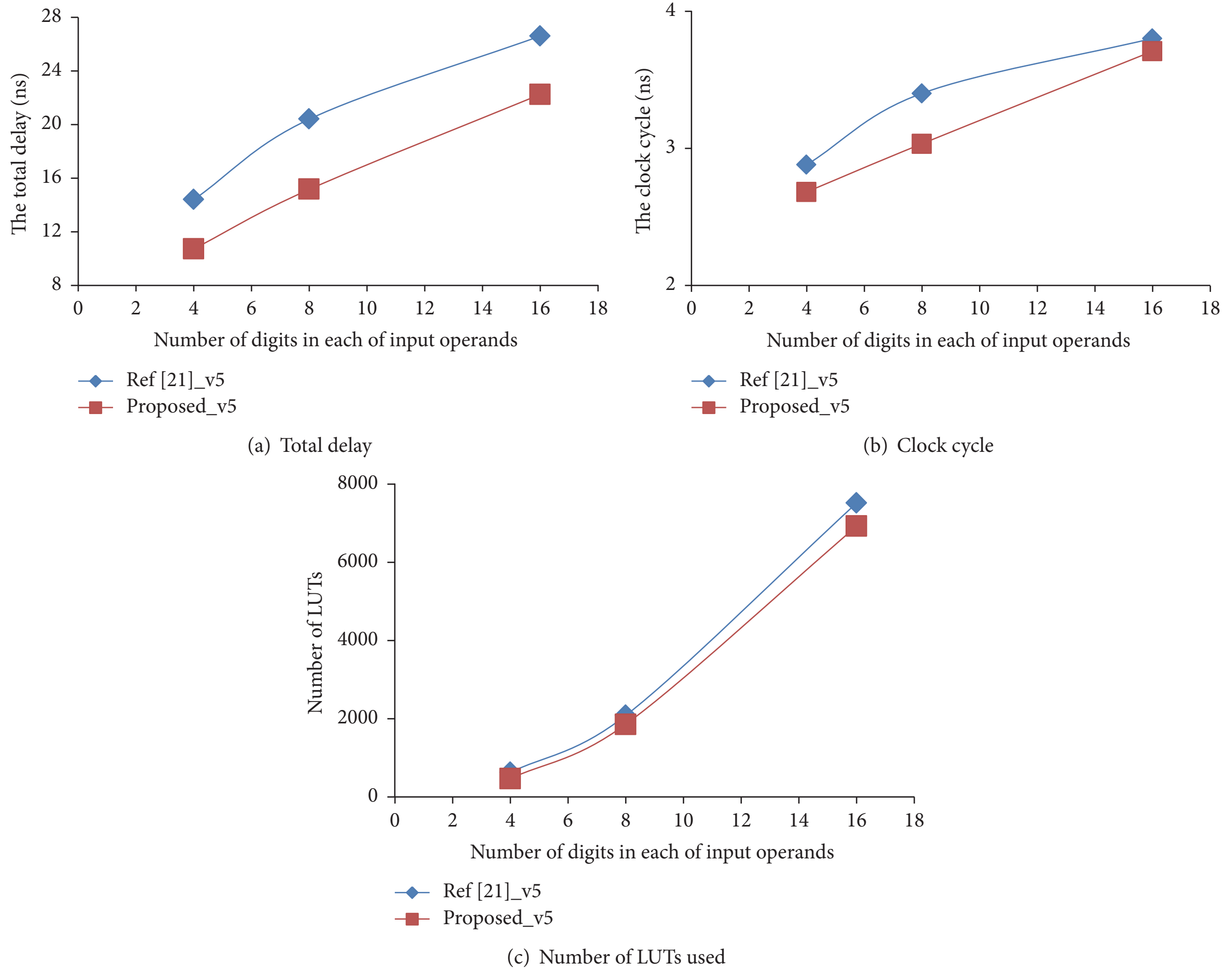


FIGURE 13: Implementation results using Virtex-5 FPGA.

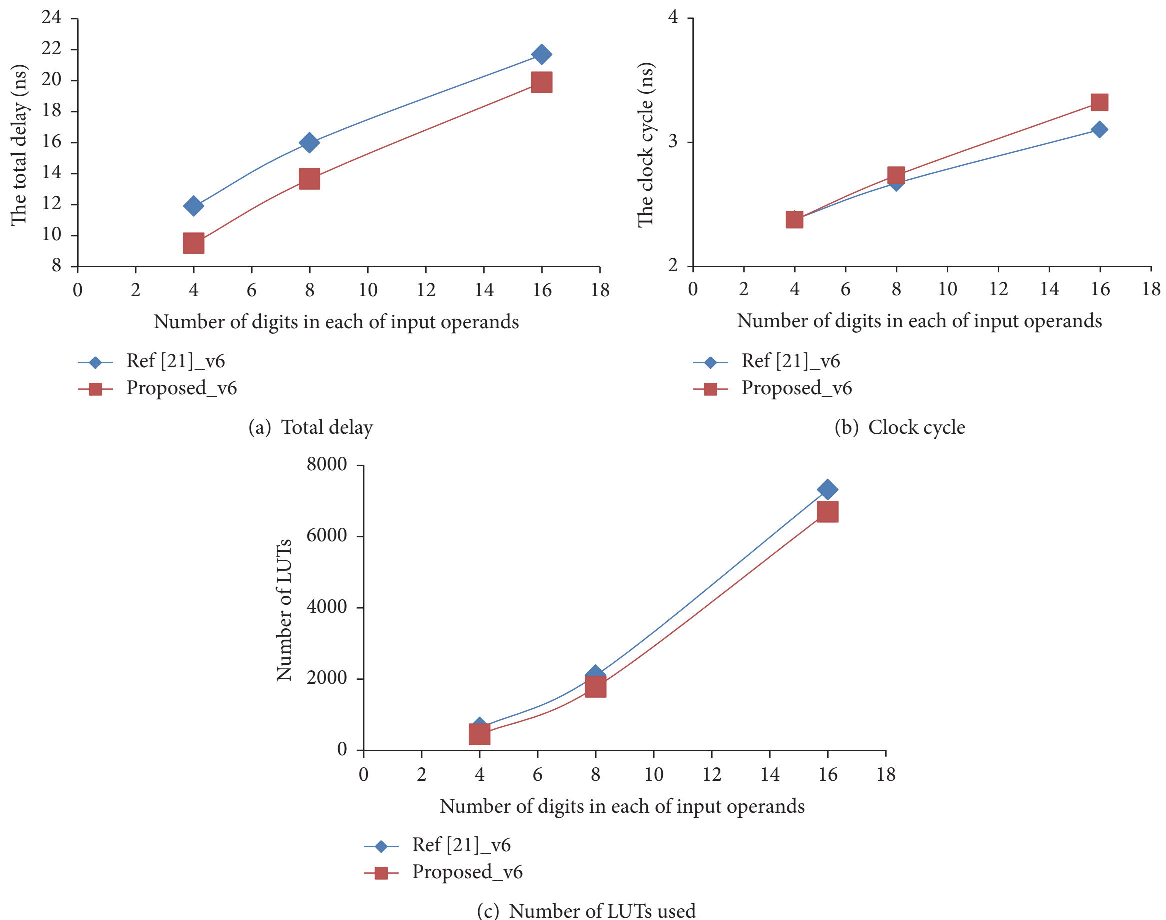
TABLE 3: Results compared with [22] for the 16 × 16-digit pipelined multiplier.

# of pipeline stages	[22]			Proposed			Comparison		
	Total delay (ns)	Clock cycle (ns)	#LUTs	Total delay (ns)	Clock cycle (ns)	#LUTs	Delay reduction (%)	Clock cycle time reduction (ns)	# of LUT saving (%)
5	27.400	5.480	6438	19.025	3.805	6843	30.57	30.57	−6.29
6	28.740	4.830	6664	22.242	3.707	6918	22.61	23.25	−3.81
7	30.660	4.460	5992	28.392	4.056	6953	7.40	9.06	−16.04

Thus, our approach compares favorably with the architectures in [21, 22]. The improvement comes in part from the use of parallel and binary operations, as well as our fast BCD additions. By using 1 × 1-digit binary multipliers and the 2-column-based binary-decimal compressors, fast parallel operations are performed with small size binary numbers. These binary-decimal compressors efficiently reduce the number of partial products to 3 or 4 decimal operands, which simplifies the decimal additions required by the multiplication. Moreover, in the decimal addition, our fast BCD adder decreases the propagation delay for BCD ripple adders. All these lead to superior multiplier architecture.

5. Conclusions

In this paper, a new $n \times n$-digit BCD multiplier approach was proposed. This approach uses 1 × 1-digit binary multipliers for the partial product generation. 2-digit column-based binary operations are used for partial product reduction. This proposed binary-decimal compression scheme makes



(a) Total delay

(b) Clock cycle

(c) Number of LUTs used

Figure 14: Implementation results using Virtex-6 FPGA.

efficient use of a parallel strategy and of fast binary operation schemes to reduce the number of partial products of the multiplier. After the binary-decimal compression, only 3 or 4 operands in general need to be added in decimal to receive the final result of a BCD multiplier. To perform the decimal additions, a fast 6-LUTs-based BCD adder was proposed to realize BCD ripple adders required for the multiplication. The proposed BCD multipliers were pipelined and implemented on Xilinx Virtex-5 and Virtex-6 FPGAs. Compared with existing architectures, improved results have been achieved.

Competing Interests

The authors declare that they have no competing interests.

References

[1] IEEE Computer Society, "IEEE 754-2008 Standard for Floating-Point Arithmetic," August 2008 http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4610935.

[2] B. Hickmann, M. Schulte, and M. Erle, "Improved combined Binary/Decimal Fixed-Point multipliers," in *Proceedings of the 26th IEEE International Conference on Computer Design (ICCD '08)*, pp. 87–94, Lake Tahoe, Calif, USA, October 2008.

[3] R. D. Kenney, M. J. Schulte, and M. A. Erle, "A high-frequency decimal multiplier," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD '04)*, pp. 26–29, October 2004.

[4] J. Bhattacharya, A. Gupta, and A. Singh, "A high performance Binary to BCD converter for decimal multiplication," in *Proceedings of the IEEE International Symposium on VLSI Design Automation and Test (VLSI-DAT '10)*, pp. 315–318, Hyderabad, India, 2010.

[5] G. Jaberipur and A. Kaivani, "Binary-coded decimal digit multipliers," *IET Computers and Digital Techniques*, vol. 1, no. 4, pp. 377–381, 2007.

[6] G. Jaberipur and A. Kaivani, "Improving the speed of parallel decimal multiplication," *IEEE Transactions on Computers*, vol. 58, no. 11, pp. 1539–1552, 2009.

[7] T. Lang and A. Nannarelli, "A radix-10 combinational multiplier," in *Proceedings of the 40th Asilomar Conference on Signals, Systems, and Computers (ACSSC '06)*, pp. 313–317, November 2006.

[8] A. Vázquez, E. Antelo, and P. Montuschi, "A new family of high—performance parallel decimal multipliers," in *Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, pp. 195–204, Montpellier, France, June 2007.

[9] A. Vázquez, E. Antelo, and P. Montuschi, "Improved design of high-performance parallel decimal multipliers," *IEEE Transactions on Computers*, vol. 59, no. 5, pp. 679–693, 2010.

[10] A. Vazquez, E. Antelo, and J. D. Bruguera, “Fast radix-10 multiplication using redundant BCD codes,” *IEEE Transactions on Computers*, vol. 63, no. 8, pp. 1902–1914, 2014.

[11] A. Kaivani, L. Han, and S.-B. Ko, “Improved design of high-frequency sequential decimal multipliers,” *Electronics Letters*, vol. 50, no. 7, pp. 558–560, 2014.

[12] M. Zhu, A. M. Baker, and Y. Jiang, “On a parallel decimal multiplier based on hybrid 8421–5421 BCD recoding,” in *Proceedings of the IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS ’13)*, pp. 1391–1394, IEEE, Columbus, Ohio, USA, August 2013.

[13] M. Zhu and Y. Jiang, “An area-time efficient architecture for 16 x 16 decimal multiplications,” in *Proceedings of the 10th International Conference on Information Technology: New Generations (ITNG ’13)*, pp. 210–216, April 2013.

[14] L. Han and S.-B. Ko, “High-speed parallel decimal multiplication with redundant internal encodings,” *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 62, no. 5, pp. 956–968, 2013.

[15] H. C. Neto and M. P. Véstias, “Decimal multiplier on FPGA using embedded binary multipliers,” in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL ’08)*, pp. 197–202, September 2008.

[16] R. K. James, K. P. Jacob, and S. Sasi, “Performance analysis of double digit decimal multiplier on various FPGA logic families,” in *Proceedings of the 5th Southern Conference on Programmable Logic (SPL ’09)*, pp. 165–170, IEEE, São Carlos, Brazil, April 2009.

[17] O. D. Al-Khaleel, N. H. Tulić, and K. M. Mhaidat, “FPGA implementation of binary coded decimal digit adders and multipliers,” in *Proceedings of the 8th International Symposium on Mechatronics and its Applications (ISMA ’12)*, Sharjah, United Arab Emirates, April 2012.

[18] G. Sutter, E. Todorovich, G. Bioul, M. Vazquez, and J.-P. Deschamps, “FPGA implementations of BCD multipliers,” in *Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig ’09)*, pp. 36–41, Quintana Roo, Mexico, December 2009.

[19] Á. Vázquez and F. De Dinechin, “Efficient implementation of parallel BCD multiplication in LUT-6 FPGAs,” in *Proceedings of the International Conference on Field-Programmable Technology (FPT ’10)*, pp. 126–133, December 2010.

[20] M. Véstias and H. Neto, “Parallel decimal multipliers and squarers using Karatsuba-Ofman’s algorithm,” in *Proceedings of the 15th Euromicro Conference on Digital System Design (DSD ’12)*, pp. 782–788, Izmir, Turkey, September 2012.

[21] C. E. M. Guardia, “Implementation of a fully pipelined BCD multiplier in FPGA,” in *Proceedings of the 8th Southern Programmable Logic Conference (SPL ’12)*, pp. 1–6, March 2012.

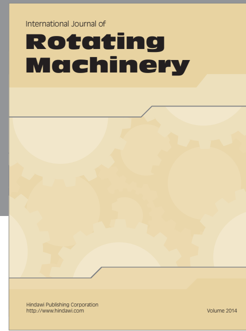
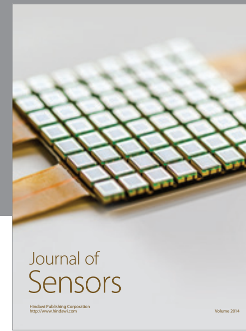
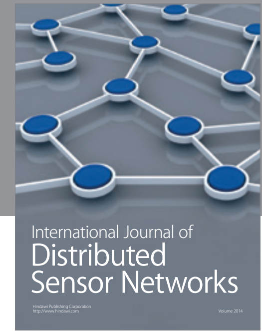
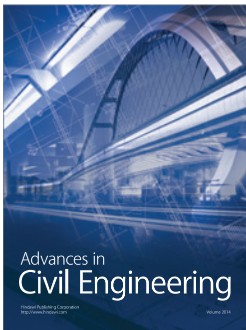
[22] M. Baesler, S.-O. Voigt, and T. Teufel, “An IEEE 754-2008 decimal parallel and pipelined FPGA floating-point multiplier,” in *Proceedings of the 20th International Conference on Field Programmable Logic and Applications (FPL ’10)*, pp. 489–495, Milano, Italy, September 2010.

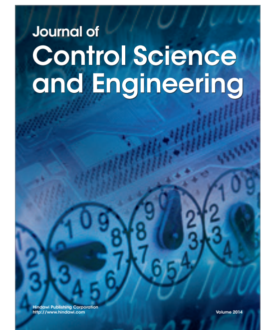
[23] Xilinx Inc, “Virtex-6 User Guide,” February 2012, http://www.xilinx.com/support/documentation/user_guides/ug364.pdf.

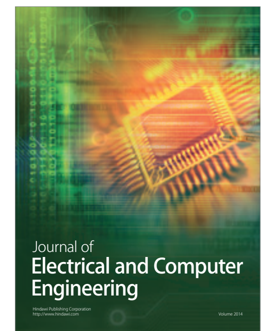
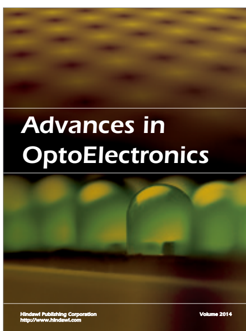
[24] S. Gao, D. Al-Khalili, and N. Chabini, “FPGA realization of high performance large size computational functions: multipliers and applications,” *Analog Integrated Circuits and Signal Processing*, vol. 70, no. 2, pp. 165–179, 2012.

[25] Binary-to-BCD Converter, Double-Dabble Binary-to-BCD Conversion Algorithm, http://www.tkt.cs.tut.fi/kurssit/1426/S12/Ex/ex4/Binary2BCD.pdf.

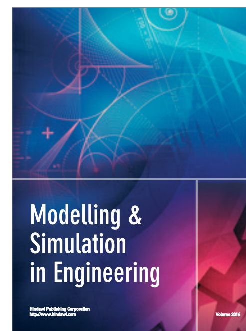
Journal of
Engineering

The Scientific World Journal

International Journal of
Rotating Machinery

Journal of
Sensors

International Journal of
Distributed Sensor Networks

Advances in
Civil Engineering

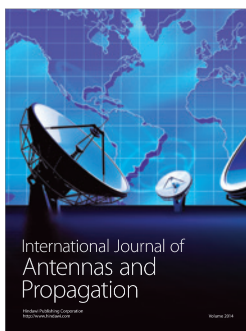
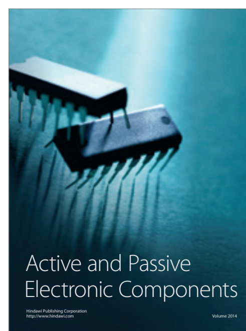
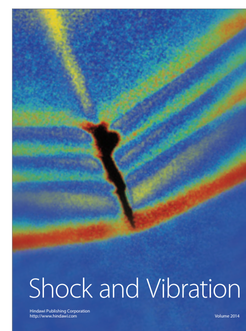
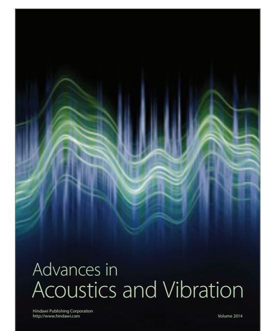
Hindawi
Submit your manuscripts at
https://www.hindawi.com

Journal of
Control Science and Engineering

Journal of
Robotics

Journal of
Electrical and Computer Engineering

Advances in
OptoElectronics

VLSI Design

International Journal of
Navigation and Observation

Modelling & Simulation in Engineering

International Journal of
Aerospace Engineering

International Journal of
Chemical Engineering

International Journal of
Antennas and Propagation

Active and Passive Electronic Components

Shock and Vibration

Advances in
Acoustics and Vibration