Digital Signal Processing 62 (2017) 1–10

Contents lists available at ScienceDirect

Digital Signal Processing

www.elsevier.com/locate/dsp

A new area-efficient BCD-digit multiplier

Encarnación Castillo [a,*], Antonio Lloris [a], Diego P. Morales [a], Luis Parrilla [a], Antonio García [a], Guillermo Botella [b]

[a] Department of Electronics and Computer Technology, Campus Universitario Fuentenueva, University of Granada, 18071 Granada, Spain
[b] Department of Computer Architecture and Automation, Complutense University of Madrid, 28040 Madrid, Spain

ARTICLE INFO

Article history:
Available online 27 October 2016

Keywords:
BCD enconding
Computer arithmetic
FPGA
IoT
Multiplier

ABSTRACT

In the Internet of Things era, with millions of devices performing financial and commercial operations, decimal arithmetic has become very popular in the computation of many business and commercial applications, in order to maintain decimal rounding and precision. This paper proposes the design and implementation of a new algorithm for decimal multiplication oriented to area reduction. This algorithm is especially suitable for field programmable gate arrays (FPGA) and has thus been implemented on this kind of devices. Results show that the proposed BCD multiplication leads to a significant area reduction without decreasing system performance.

1. Introduction

Decimal arithmetic is easily managed in human-oriented applications, with BCD encoding the dominant representation for decimal digits. However, it has had a limited use in numerical data processing [1] in the past due to the traditional drawbacks and increased area required for radix-10 digital implementations. Thus, binary arithmetic has become the most widely used for performing arithmetic and logical operations in all digital circuits due to its better numerical properties and simpler implementation in two-state digital systems. Despite the large use of binary calculations, a renewed interest on efficient decimal arithmetic implementations has emerged due to an increasing demand from many human-centric applications, such as financial trading and other business applications [2], or Internet-of-Things (IoT) computing [3]. In this sense, McKinsey Global Institute reports the IoT business will deliver $6.2 trillion of revenue by 2025 [4], so the IoT is expected to have an enormous impact on the commercial and financial aspects of our lives. The availability of small portable devices which have the power to perform large amounts of financial and commercial operations in international markets implies that this portable devices have to achieve the required precision. Many financial applications are using decimal arithmetic due to the rounding errors obtained when decimal fractions cannot be represented with a finite number of bits in binary encoding. For example, the number 0.1 has a precise decimal representation but cannot be exactly represented as a binary number with a finite number of bits. As a consequence, binary round up and truncate operations introduce errors in the final results. These errors are not acceptable for some applications, for example, financial, accounting and business applications and thus, decimal arithmetic becomes a necessity. This motivates the current intense research and development on decimal arithmetic algorithms and architectures [5–7].

The IoT devices do not use processors which have the required hardware support for decimal operations, and general purpose processors implement decimal arithmetic using time consuming software routines. The main resource for increasing speed and energy efficiency in computer systems is hardware acceleration [8–10]. Graphics Processing Units, or GPUs, and FPGA-based (Field Programmable Gate Array) accelerators have emerged as the most relevant options. Concretely, FPGA-based accelerators are very useful for optimizing the hardware required by applications demanding a specific number system, for example decimal system. In addition, FPGA are used as coprocessor in IoT devices, offering reduced cost and versatility [11]. Thus, these current technologies allow to enhance the performance of decimal computations, getting an increase of processing speed without expensive area costs. This facilitates the hardware realizations of decimal arithmetic algorithms and their implementations using the available binary technology. Besides, the revision of the IEEE 754 Standard for Floating-Point Arithmetic (IEEE 754-2008) [12] incorporates a specification of decimal floating point arithmetic (DFP) that can be implemented in software, hardware, or in a combination of both, very useful for high performance.

Decimal multiplication is an important operation frequently used in many applications. However, along with division [13,14]

* Corresponding author.
E-mail address: encas@ugr.es (E. Castillo).

http://dx.doi.org/10.1016/j.dsp.2016.10.011
1051-2004/

decimal multiplication is a basic but complex operation [15,16]. Hardware implementation of BCD multiplication has required special attention due to the larger range of decimal digits and the use of binary codes representing decimal values. In recent years, many decimal multiplication algorithms have been proposed [17–19,16,20–22]. Several designs for fixed-point decimal multiplications are proposed in [17], which are oriented to reducing the critical path delay. In [18] authors present an iterative multiplication using the ideas of doubling and quintupling, in order to reduce the number of additions and speed up the multiplication. This is achieved by easily obtaining the double and the quintuple of the multiplicand, and then using the multiplier value for picking multiples of the multiplicand from a predefined set. Some other authors [7] have used the concept of redundant codes to enhance radix-10 multipliers.

This paper presents a new algorithm providing an area-efficient decimal multiplication, and its FPGA implementation. This new proposal is based on the special characteristics of the BCD number system and the analysis of the logic operations involved in conventional decimal multipliers.

The rest of the paper is organized as follows. The new algorithm for BCD multiplication is presented in Section 2. The proposed BCD multiplier architecture is detailed in Section 3, while Section 4 introduces two architectures for binary-to-BCD conversion. Section 5 is devoted to FPGA implementation results and performance comparisons. Finally, Section 6 summarizes the main conclusions.

2. Proposed BCD-digit multiplication algorithm

Decimal multiplication can be implemented by directly working with numbers in a BCD system. Compared to binary multiplication, this option involves an important increase of the complexity. Thus, decimal multipliers are usually implemented using an iterative approach [23,24] which reduces the mentioned complexity. An alternative for implementing BCD multipliers is the 2-stage multiplication method, which is composed of a binary product stage for computing the binary multiplication, and a binary-to-BCD decoding stage for correctly converting the result back to BCD [25–27]. In this work, we have focused on the design of an efficient binary multiplier for BCD multiplication. The effective combination of the new binary multiplier with a binary-to-BCD converter makes the proposed solution competitive. The efficiency of the new binary multiplier is based on binary arithmetic taking advantage of the non-used bit streams in the binary representation of the BCD digits (from 1010 to 1111).

Let $A = a_3a_2a_1a_0$ and $B = b_3b_2b_1b_0$ be two BCD digits. The BCD-coded product consists of two BCD digits $D = d_3d_2d_1d_0$ and $C = c_3c_2c_1c_0$:

$$A \times B = D \times 10 + C \tag{1}$$

Taking into account this 2-stage multiplication method, the binary product stage computes a 7-bit binary number $P = p_6p_5p_4p_3p_2p_1p_0$ in {0,1,...81}, while the binary-to-BCD decoding stage computes the BCD code for the obtained binary result, *i.e.*, it computes D and C from P. The architecture for the conventional 4-bit multiplier is composed of 16 two-input AND gates (generating the corresponding 1×1-bit partial products), 4 Half-Adders (HA) and 8 Full-Adders (FA). This multiplier could be used for the first described stage of the decimal multiplication. However, since inputs to this multiplier will be only in the set {0, 9} it will be possible to reduce the required operations and thus the implementation resources. In the following, this idea is extended and applied to describe the proposed design of the binary multiplication stage for two BCD digits.

Require: $A = a_3a_2a_1a_0$, $B = b_3b_2b_1b_0$
Ensure: $P = p_6p_5p_4p_3p_2p_1p_0$
1: **if** $(a_3 \vee b_3 = 1)$ **then**
2: **if** $(a_3 = 1)$ **then**
3: **if** $(a_3a_2a_1a_0 = 1000)$ **then**
4: $P \longleftarrow b_3b_2b_1b_0000$
5: **end if**
6: **if** $(a_3a_2a_1a_0 = 1001)$ **then**
7: $P \longleftarrow b_3b_2b_1(b_0 + b_3)b_2b_1b_0$
8: **end if**
9: **else** $\{b_3 = 1\}$
10: **if** $(b_3b_2b_1b_0 = 1000)$ **then**
11: $P \longleftarrow a_3a_2a_1a_0000$
12: **end if**
13: **if** $(b_3b_2b_1b_0 = 1001)$ **then**
14: $P \longleftarrow a_3a_2a_1(a_0 + a_3)a_2a_1a_0$
15: **end if**
16: **end if**
17: **else** $\{a_3 \vee b_3 = 0\}$
18: $P \longleftarrow a_2a_1a_0 \times b_2b_1b_0$ {3-bit binary multiplier}
19: **end if**

Fig. 1. Binary multiplication algorithm for BCD digits.

2.1. New binary multipliers oriented to BCD-digit multiplication

The proposed algorithm for binary multiplication of two BCD digits, A and B, takes advantage of the binary representation of these digits, from 0000 to 1001. In fact some combinations of $a_3a_2a_1a_0 \times b_3b_2b_1b_0$ are never possible, for example 1100×0010. For operands A and B there are only two valid BCD digits whose most significant bit (MSB) is set to 1: 1000 and 1001. Therefore, the proposed algorithm distinguishes between two cases:

- if ($a_3 = 1$ or $b_3 = 1$): this condition corresponds to values 1000 and 1001 for operand A or operand B. This condition is equivalent to $a_3 \vee b_3 = 1$.
- if ($a_3 = 0$ and $b_3 = 0$): this condition corresponds to values from 0000 to 0111 for operand A and operand B. This condition is equivalent to $a_3 \vee b_3 = 0$.

It must be noted that, in this manuscript, symbols $\vee$ and $\wedge$ are used for Boolean operators OR and AND, respectively, while + corresponds to arithmetic addition (which could generate carry propagations). Fig. 1 describes the proposed binary multiplier. As it can be observed, for the case $(a_3 \vee b_3 = 1)$, the 7-bit result depends on operand $a_3a_2a_1a_0$ or operand $b_3b_2b_1b_0$. If $a_3a_2a_1a_0 = 1000$, the result is equivalent to a 3-bit left shift of operand B, filling with 0s (line 4 in Fig. 1), resulting in $P = b_3b_2b_1b_0000$. On the other hand, if $a_3a_2a_1a_0 = 1001$, the result is equivalent to a 3-bit left shift of operand B, filling with $b_2b_1b_0$, plus an addition in the fourth LSB bit, $b_0 + b_3$, thus obtaining $P = b_3b_2b_1(b_0 + b_3)b_2b_1b_0$ (line 7 in Fig. 1). A similar reasoning is made for the development of the algorithm for $b_3b_2b_1b_0 = 1000$ and $b_3b_2b_1b_0 = 1001$ (lines 11 and 14 in Fig. 1). For the second case, $a_3 = 0$ and $b_3 = 0$, the multiplication can be obtained using a 3-bit binary multiplier (line 18 in Fig. 1).

In order to reduce the complexity of the proposed algorithm, some modifications can be introduced, thus obtaining the algorithm described in Fig. 2. Line 6 in Fig. 1 reads "if $a_3a_2a_1a_0 = 1001$", so if the condition $b_3b_2b_1b_0 \neq 1001$ is added, arithmetic addition in line 7 could be converted into a logic addition. This modification is possible since, with this new condition ($b_3b_2b_1b_0 \neq 1001$), only one of the adding bits is 1, b_0 or b_3, thus the arithmetic addition can be replaced by an OR operation, resulting $P = b_3b_2b_1(b_0 \vee b_3)b_2b_1b_0$ (line 7 in Fig. 2). This modification will save resources, while it also implies there is no carry propagation, sav-

Require: $A = a_3a_2a_1a_0$, $B = b_3b_2b_1b_0$
Ensure: $P = p_6p_5p_4p_3p_2p_1p_0$
1: **if** $(a_3 \vee b_3) = 1$ **then**
2: **if** $(a_3 = 1)$ **then**
3: **if** $(a_3a_2a_1a_0 = 1000)$ **then**
4: $P \longleftarrow b_3b_2b_1b_0000$
5: **end if**
6: **if** $(a_3a_2a_1a_0 = 1001)$ **then**
7: $P \longleftarrow b_3b_2b_1(b_0 \vee b_3)b_2b_1b_0$
8: **end if**
9: **else** $\{b_3 = 1\}$
10: **if** $(b_3b_2b_1b_0 = 1000)$ **then**
11: $P \longleftarrow a_3a_2a_1a_0000$
12: **end if**
13: **if** $(b_3b_2b_1b_0 = 1001)$ **then**
14: $P \longleftarrow a_3a_2a_1(a_0 \vee a_3)a_2a_1a_0$
15: **end if**
16: **end if**
17: **else** $\{a_3 \vee b_3 = 0\}$
18: $P \longleftarrow a_2a_1a_0 \times b_2b_1b_0$ {3-bit binary multiplier}
19: **end if**
20: **if** $(a_3a_2a_1a_0 = 1001)$ and $(b_3b_2b_1b_0 = 1001)$ **then**
21: $P \longleftarrow p_6p_510p_2p_1p_0$ {fixes p_4 to 1 and p_3 to 0 for correcting the result}
22: **end if**

Fig. 2. Modified binary multiplication algorithm for BCD digits.

ing also the related logic. The same type of modifications can be introduced for operands $b_3b_2b_1b_0$ corresponding to lines 13 and 14 in Fig. 1. On the other hand, introducing these modifications, the result of the multiplication 1001×1001 is not correct. Since the addition operation was changed into an OR operation, the result obtained for 1001×1001 is $1001001 = 73$, while the correct one is $1010001 = 81$. Comparing both data, we can see that only p_4 and p_3 are wrong. Thus, interchanging the value of these bits or fixing the values $p_4 = 1$ and $p_3 = 0$ make the result correct (line 21 in Fig. 2).

The architecture implementing this new algorithm is presented in next section.

3. Proposed binary multiplication architecture for BCD multiplier

The proposed general architecture for the modified algorithm described in Fig. 2 is depicted in Fig. 3. This figure includes three main blocks, called "Circuit $(a_3 \vee b_3) = 1$", "3×3-bit multiplier", and "Correction of 1001×1001". These blocks are detailed in the following.

3.1. Circuit $(a_3 \vee b_3) = 1$

Fig. 4 shows a possible architecture for this circuit. As it can be seen in this figure, if $a_3 = 1$ the multiplexer outputs are connected to the second input, thus:

$p'_6 = b_3$
$p'_5 = b_2$
$p'_4 = b_1$
$p'_3 = (\bar{a}_0 \wedge b_0) \vee (a_0 \wedge (b_3 \vee b_0)$
$p'_2 = b_2 \wedge a_0$
$p'_1 = b_1 \wedge a_0$
$p'_0 = b_0 \wedge a_0$

As it can be observed, p'_3, p'_2, p'_1 and p'_0 depend on a_0 for providing the correct result. It is easily proved that these logic expressions are related to lines 4 and 7 in Fig. 2. Similarly, if $a_3 = 0$ (but $b_3 = 1$, since $(a_3 \vee b_3) = 1$) the logic expressions obtained

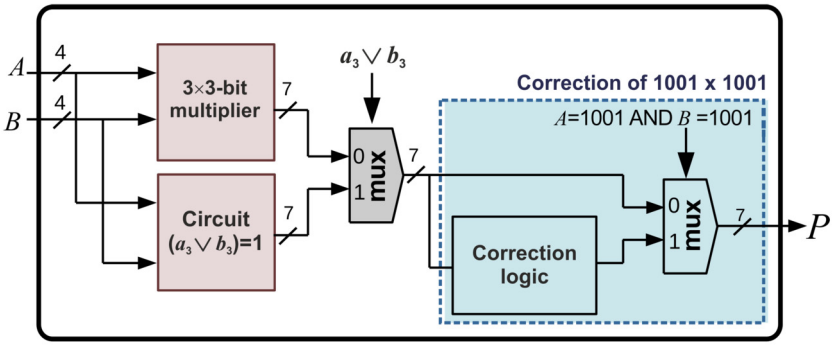


Fig. 3. General architecture for the proposed binary multiplier of two BCD digits according to the modified algorithm.

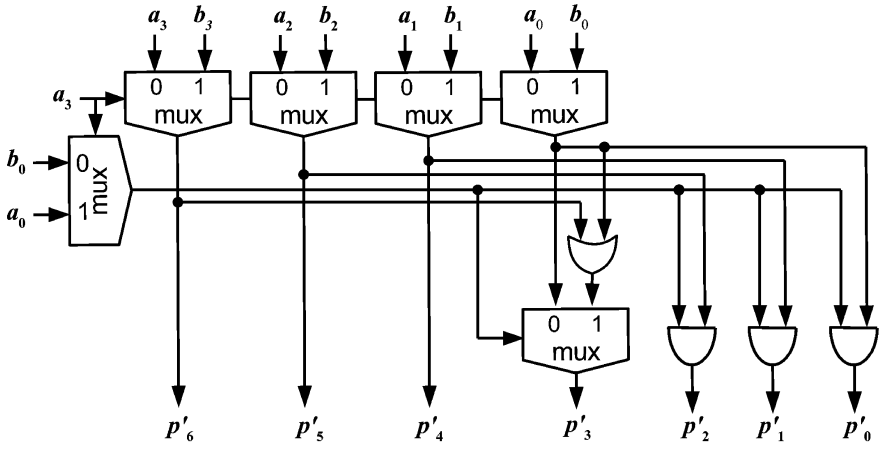


Fig. 4. Architecture for "Circuit $(a_3 \vee b_3) = 1$".

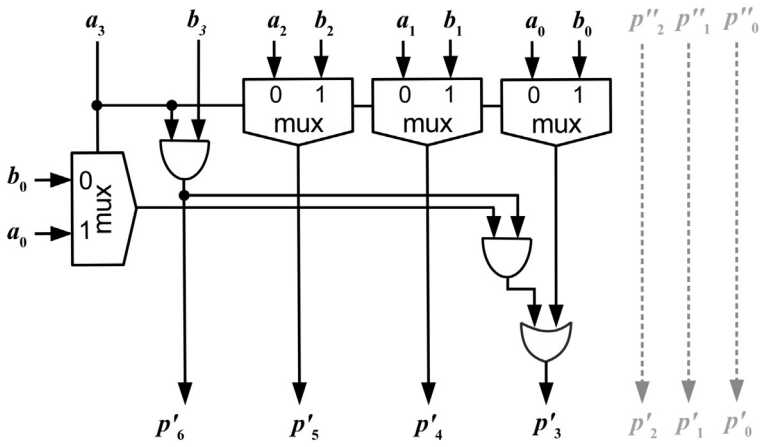


Fig. 5. Enhanced architecture for "Circuit $(a_3 \vee b_3) = 1$".

from the analysis of this circuit are related to lines 11 and 14 in Fig. 2. This architecture can be optimized in order to reduce the required logic gates. Thus, an enhanced version of this architecture is shown in Fig. 5. The modifications are:

- The multiplexer with output p'_6 is replaced by an AND gate, since $p'_6 = a_3 \wedge b_3$.
- Let us consider the logic involved by the OR gate followed by the multiplexer with the output p'_3:
  - For $a_3 = 1$:
    $p'_3 = (\bar{a}_0 \wedge b_0) \vee (a_0 \wedge (b_3 \vee b_0) = b_0 \vee (a_0 \wedge b_3)$
  - For $a_3 = 0$:
    $p'_3 = (\bar{b}_0 \wedge a_0) \vee (b_0 \wedge (a_3 \vee a_0) = a_0 \vee (b_0 \wedge a_3)$

  Thus, the OR gate and the multiplexer are replaced by an AND gate and an OR gate, as it is shown in Fig. 5.
- The outputs p'_2, p'_1 and p'_0 are equal to outputs p''_2, p''_1 and p''_0 of the "3 × 3-bit multiplier" block, which will be detailed in next subsection. Thus, outputs p'_2, p'_1 and p'_0 can be removed to avoid redundant logic in the overall circuit and to save resources.

3.2. Implementation of the 3 × 3-bit multiplier

As Fig. 3 shows, if $a_3 \vee b_3 = 0$ the result generated by the 3 × 3-bit multiplier will be directly fed to the output of the 7-bit binary multiplier. Thus, the product for this case is $A \times B = a_2a_1a_0 \times b_2b_1b_0$. There are different options for implementing the 3 × 3-bit multiplier. Three different architectures are proposed:

1. $Mul(3 \times 3)_1$. This option corresponds to the classical 3 × 3-bit multiplier based on the partial products and the required HAs and FAs.
2. $Mul(3 \times 3)_2$. This is a new proposal for the 3 × 3-bit multiplier based on a 2 × 2-bit multiplier, the partial products depending on a_2 and b_2, and the required HAs and FAs. The architecture of the 2 × 2-bit multiplier is shown in Fig. 6, where m_i functions, for $i = 0, 1, 2, 3$, are based on the following logic expressions:
   $m_0 = a_0 \wedge b_0$
   $m_1 = (a_0 \wedge b_1 \wedge (\bar{a}_1 \vee \bar{b}_0)) \vee (a_1 \wedge b_0 \wedge (\bar{a}_0 \vee \bar{b}_1))$
   $m_2 = a_1 \wedge b_1 \wedge (\bar{a}_0 \vee \bar{b}_0)$
   $m_3 = a_1 \wedge a_0 \wedge b_1 \wedge b_0$
3. $Mul(3 \times 3)_3$. This option for the multiplier consists of a high-level description of the 3×3-bit multiplication obtained from direct transcription of its truth table. In this case, synthesis tools will carry out logic optimizations.

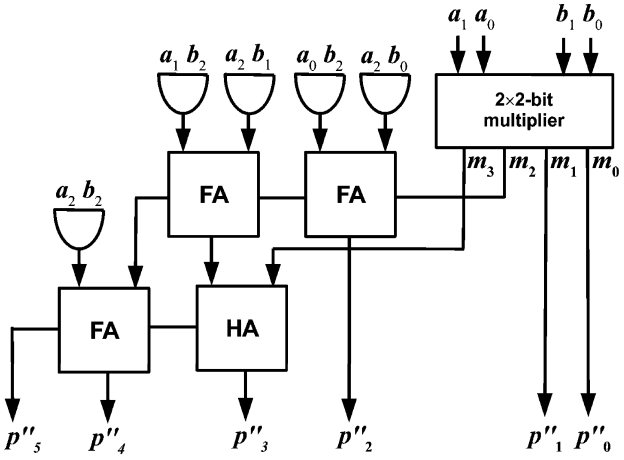


Fig. 6. Second architecture for the 3 × 3-bit multiplier.

3.3. Data Driving Logic (DDL)

The architecture for this new binary multiplier includes the multiplexer driven by the signal $(a_3 \vee b_3)$ which selects the correct multiplication data, as it is shown in Fig. 3. It is important to note that after analyzing the design, only 3 bits $(p_5p_4p_3)$ of the 7-bit global output are driven by this multiplexer, since the other 4 bits (p_6 and $p_2p_1p_0$) can be directly connected to the output signal p'_6 of the block "Circuit $(a_3 \vee b_3) = 1$", and to output signals $p''_2p''_1p''_0$ of the block "3 × 3-bit multiplier", respectively. Taking into account that if $(a_3 \vee b_3) = 1$, the only possible values for A or/and B are 1000 and 1001, it is possible to reduce the logic required by the conventional multiplexer architecture. Thus, it is not necessary to include full multiplexers in the design, and the logic can be reduced to the arrangement shown in Fig. 7 as "Data Driving Logic".

3.4. Correction of **1001** × **1001** ($C_{9\times9}$)

As it was detailed in Section 2, for the multiplication 1001 × 1001 a correction is necessary. For inputs $a_3a_2a_1a_0 = b_3b_2b_1b_0 =$ 1001, the block "Circuit $(a_3 \vee b_3) = 1$" generates the output $P' =$ 1001001 = 73, while the correct output is 1010001 = 81. Comparing the data 1001001 = 73 and 1010001 = 81, it can be observed that two bits are wrong, namely p'_4 and p'_3. After evaluating several possibilities which reduce as much as possible the required logic, the correction has been made detecting the input values $(A = B = 1001)$ and, for this case, fixing p_4 to 1 and p_3

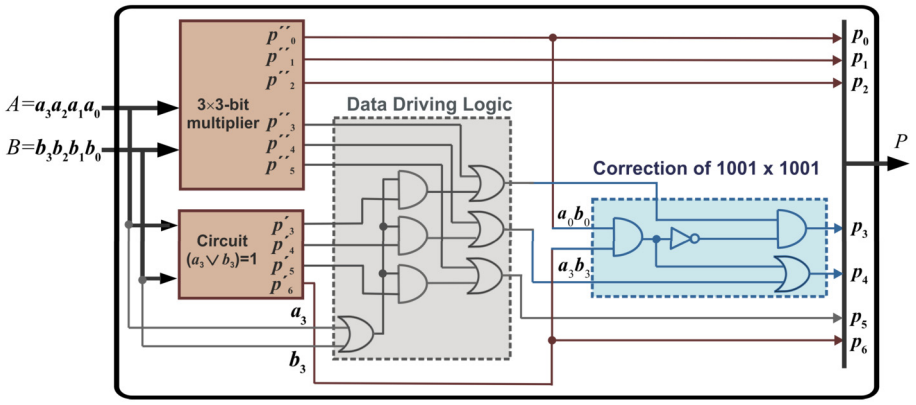


Fig. 7. Specific architecture for the proposed binary multiplier of two BCD digits.

to 0, as it was indicated in line 21 in Fig. 2. For the detection of $a_3a_2a_1a_0 = b_3b_2b_1b_0 = 1001$, a 2-input AND gate is used, whose inputs are $p''_0 = a_0 \wedge b_0$, obtained from the "3 × 3-bit multiplier" block, and $p'_6 = a_3 \wedge b_3$ obtained from the "Circuit $(a_3 \vee b_3) = 1$" block. The architecture proposed for this correction is shown in Fig. 7.

4. Binary-to-BCD converters

As it was commented in Section 2, the two-stage BCD-digit multiplication approach requires a second stage to convert 7-bit binary data to two 4-bit BCD digits. In the following, two different binary-to-BCD converters are presented.

4.1. Converter$_1$

The design for this binary-to-BCD converter is based on a high-level description of this truth table including some modifications. Let us remember again that the data to be converted is $P = p_6p_5p_4p_3p_2p_1p_0$, *i.e.*, the 7-bit binary multiplication output of two BCD digits ($a_3a_2a_1a_0$ and $b_3b_2b_1b_0$). The converter will return two BCD digits, $D = d_3d_2d_1d_0$ and $C = c_3c_2c_1c_0$. For this converter, the corresponding truth table has 38 different input patterns (corresponding to the number of different 7-bit binary multiplication outputs), thus the remaining 90 input symbols will be assigned a "don't care" output. This makes possible the reduction of the related logic expressions. In addition, considering the logic expression in $c_0 = a_0 \wedge b_0 = p_0$, the p_0 output of the converter can be implemented simply connecting c_0 to p_0. On the other hand, $d_3 = a_3 \wedge b_3 \wedge a_0 \wedge b_0 = p_6 \wedge p_0$, thus d_3 can be obtained directly using the bits of the binary product p_6 and p_0 and one AND gate. From the previous analysis and in order to reduce the area resources, we propose an architecture for the converter, called $Converter_1$, based on the following description:

1. A 7-bit input ($p_6p_5p_4p_3p_2p_1p_0$), 6-bit output ($d_2d_1d_0$ and $c_3c_2c_1$) truth table with 90 "don't care" output symbols.
2. The following logic expressions for c_0 and d_3:
   $c_0 = p_0$
   $d_3 = p_6 \wedge p_0$

4.2. Converter$_2$

For this proposed converter, outputs $d_2d_1d_0$ and $c_3c_2c_1$ are implemented using the logic expressions of one of the binary-to-BCD conversions proposed by Sutter et al. [28], which they called Arithmetic II, while the outputs c_0 and d_3 are implemented using the same expressions described above for $Converter_1$. Thus, the logic expressions used for this converter are:

$c_3c_2c_1 = cc_3cc_2cc_1 + cy_1(cy_1 \vee cy_0)cy_0$
$c_0 = p_0$
$d_2d_1d_0 = dd_2dd_1dd_0 + 0cy_1cy_0$
$d_3 = p_6 \wedge p_0$
where dd, cy and cc are the same signals as in [28].

5. Implementation results and discussions

In this section we have evaluated our proposal for BCD-digit multiplication. First, we have evaluated the proposed binary multipliers, followed by the BCD-digit multipliers including these binary multipliers and the additional converters.

5.1. Evaluation of binary multipliers

The proposed binary multipliers to be evaluated are called $Proposed_1$, $Proposed_2$ and $Proposed_3$, based on the structure of Fig. 7 and including the $Mul(3\times3)_1$, $Mul(3\times3)_2$ and $Mul(3\times3)_3$, respectively. Thus, each one of these multipliers includes the "3 × 3-bit multiplier" block the "Circuit $(a_3 \vee b_3) = 1$" block, shown in Fig. 5, the logic for the Correction of 1001 × 1001, $C_{9\times9}$, detailed in Fig. 7, and the "Data Driving Logic", also detailed in Fig. 7. The proposed binary multipliers for BCD digits, $Proposed_1$, $Proposed_2$ and $Proposed_3$, were evaluated by comparing them to the following binary multiplier architectures:

- *Structural*. This architecture corresponds to a conventional 4 × 4-bit binary multiplier which is composed of AND gates for obtaining the partial products, $a_i \wedge b_j$ for $i, j = 0, 1, 2, 3$, and the required HAs and FAs. This multiplier is the one requiring more resources since the potentialities of BCD number system (bit string 1010 to 1111 are not used) are not exploited.
- *TruthTable*. The architecture of this design is based on the high-level description of a binary multiplier for BCD digits from the corresponding truth table. It is important to remember that this truth table has 156 "don't care" output symbols, which simplify the relevant Boolean expressions.
- *Jaberipur*07. This architecture corresponds to one of the binary multipliers proposed by Jaberipur et al. [29], concretely, the area-optimized binary product BCD-digit multiplier.
- *CorrectedFazlali*15. The binary multiplier proposed by Fazlali et al. [22] is one of the latest proposal found in the literature. This architecture is mainly based on the prediction of the carries. It has to be mentioned that some mistakes in the logic expressions of this paper were detected and changes were required to correct this binary multiplier as it is detailed in [30].

VHDL descriptions of the binary multipliers have been implemented using Spartan 6 xc6slx45t-3fgg481 and Virtex 6

Table 1
Area and delay figures for implementation of binary multipliers and BCD-digit multipliers.

Design	Architecture	Spartan 6/Virtex 6		Spartan 6		Virtex 6	
		Area		Delay		Delay	
		6-input LUTs	Ratio	(ns)	Ratio	(ns)	Ratio
Binary multiplier	$Proposed_1$	9	1.13	1.909	1.06	1.310	1.17
	$Proposed_2$	11	1.38	3.130	1 74	2.168	1.94
	$Proposed_3$	8	1.00	1.909	1.06	1.310	1.17
	$Structural$	15	1.88	4.150	2.31	2.700	2.42
	$TruthTable$	13	1.63	1.795	1.00	1.116	1.00
	$Jaberipur07$ [29]	13	1.63	4.348	2.42	3.028	2.71
	$CorrectedFazlali15$ [22,30]	13	1.63	3.035	1.69	2.134	1.91
BCD-digit multiplier	$Proposed_1 + Converter_1$	14	1.00	3.221	1.78	2.189	3.65
	$Proposed_2 + Converter_1$	17	1.21	4.442	2.45	3.047	5.09
	$Proposed_3 + Converter_1$	14	1.00	3.221	1.78	2.189	3.65
	$Structural + Converter_1$	20	1.43	5.454	3.01	3.580	5.98
	$TruthTable + Converter_1$	20	1.43	3.107	1.71	1.995	3.33
	$Jaberipur07$ [29] $+Converter_1$	28	2.00	3.264	1.80	2.300	3.84
	$CorrectedFazlali15$ [22,30] $+Converter_1$	19	1.36	4.479	2.47	3.942	6.58
	$DirectTruthTable$	22	1.57	1.926	1.06	1.136	1.90
	$Jaberipur09$ [16]	35	2.50	7.649	4.22	5.120	8.55
	$Gorgin14$ [21]	24	1.71	1.812	1.00	0.599	1.00

Table 2
Area and delay figures for implementation of binary multipliers and BCD-digit multipliers.

Design	Architecture	Virtex 5			
		Area		Delay	
		6-input LUTs	Ratio	(ns)	Ratio
Binary multiplier	$Proposed_1$	9	1.13	1.175	1.23
	$Proposed_2$	12	1.50	2.205	2.31
	$Proposed_3$	8	1.00	0.956	1.00
BCD-digit multiplier	$Proposed_1 + Converter_2$	16	1.07	1.891	3.12
	$Proposed_2 + Converter_2$	20	1.33	1.829	3.01
	$Proposed_3 + Converter_2$	15	1.00	1.911	3.15
	$Jaberipur07$ [29]	34	2.27	4.966	8.18
	$James08$ [31]	38	2.53	3.579	5.90
	$Veeramachaneni08$ [32]	28	1.87	4.015	6.61
	$Bhattacharya10$ [25]	30	2.00	4.195	6.91
	$Khaleel11$ [26]	25	1.67	3.556	5.86
	$SimplifiedKhaleel11$ [26]	25	1.67	3.455	5.69
	$Khaleel12$ [27]	25	1.67	1.337	2.20
	$NewGorgin14_1$ [21]	28	1.87	3.864	6.37
	$NewGorgin14_2$ [21]	25	1.67	3.455	5.69
	$NewGorgin14_3$ [21]	25	1.67	0.607	1.00

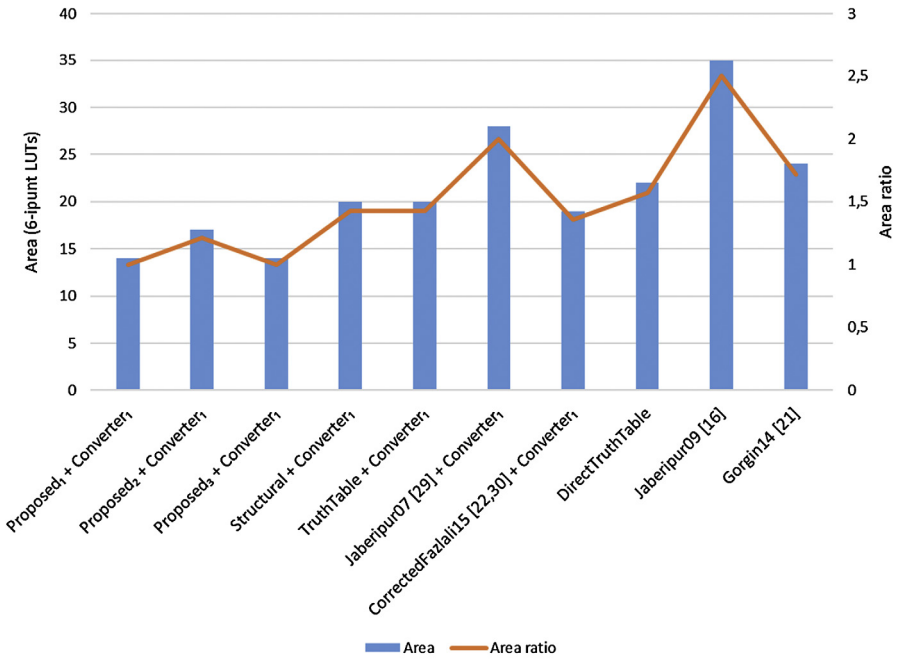
xc6vls240t-1ff1156 devices. These binary multipliers were verified through an exhaustive test. Table 1 summarizes the area and delay figures for these FPGA implementations. The included delay corresponds exclusively to the logic and its internal routing, without including those related to input and output buffers and thus excluding any pad delays. $TruthTable$ is the architecture with the smallest delays, followed closely by $Proposed_1$ and $Proposed_3$. It can be also observed that for both devices, $Proposed_1$, $Proposed_2$ and $Proposed_3$ binary multipliers occupy fewer LUTs than the other binary multipliers used for comparison. Concretely, for both devices $Proposed_3$ multiplier occupies eight 6-input LUTs, 38.46% less area than $Jaberipur07$ and $CorrectedFazlali15$. This area reduction does not deteriorate these architecture delays. Concretely, the $Proposed_3$ delay is 37.10% less than $CorrectedFazlali15$ delay using Spartan 6 device. Moreover, using Virtex 6 device, the $Proposed_3$ delay is 8.61% less than obtained with $CorrectedFazlali15$.

5.2. Evaluation of BCD-digit multipliers

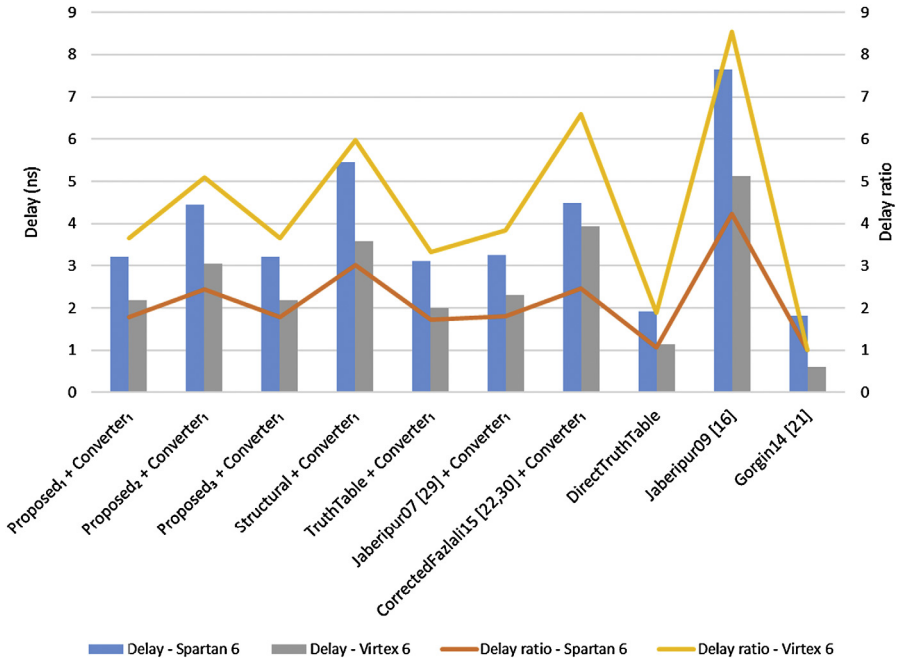
After that, we have evaluated BCD-digit multipliers based on the proposed binary multipliers. These BCD-digit multipliers are called $Proposed_1 + Converter_1$, $Proposed_2 + Converter_1$, $Proposed_3 + Converter_1$, $Structural + Converter_1$, $TruthTable + Converter_1$, $Jaberipur07 + Converter_1$ and $Corrected\text{ -}Fazlali15 + Converter_1$. They are composed of the binary multipliers evaluated in the previous subsection and the $Converter_1$ block. We have selected $Converter_1$ because it results in better area and delay figures for Spartan 6 and Virtex 6 devices than $Converter_2$. The BCD-digit multipliers were also verified through an exhaustive test. Table 1 also summarizes the delay and area figures of the different BCD-digit multipliers for the previously mentioned devices. The best area results for both Xilinx devices are obtained for $Proposed_1 + Converter_1$ and $Proposed_3 + Converter_1$, both designs occupying fourteen 6-input LUTs. Compared to $CorrectedFazlali15 + Converter_1$, these two proposed BCD multipliers get an area reduction of 26.32%. Regarding delay, the $TruthTable + Converter_3$ gets best figures for both devices, with $Proposed_1 + Converter_1$ and $Proposed_1 + Converter_1$ closer to these delays (3.70% delay increase).

On the other hand, in order to compare to other independent proposals, we have implemented the following BCD-digit multiplier architectures:

- $DirectTruthTable$. This architecture is based on the direct BCD multiplication, not needing the two stages based on binary multiplication and binary-to-BCD conversion. The ideas for this



(a) Area comparison



(b) Delay comparison

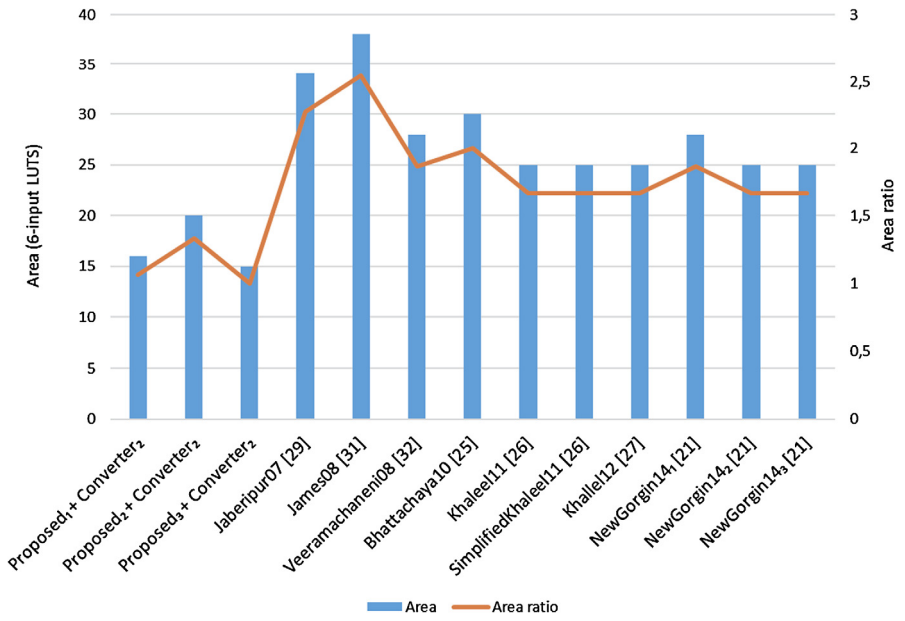
Fig. 8. Comparison of implementation results for BCD-digit multipliers in Table 1.

BCD multiplier are similar to the one proposed by Khaleel et al. [27] but the architecture proposed in the present work is described by means of a truth table in VHDL, leaving the optimization in hands of the synthesis tool.

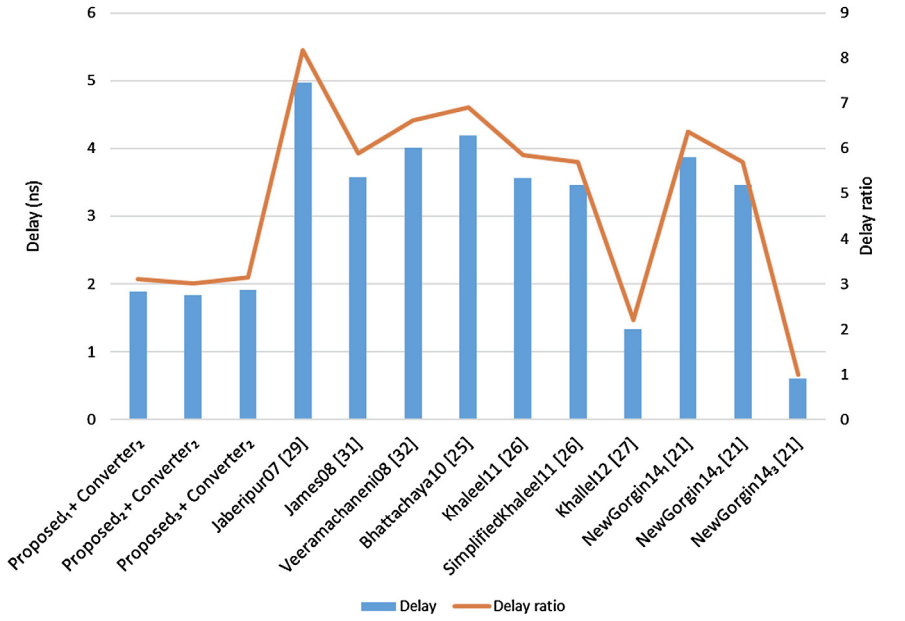
- *Jaberipur*09. This architecture was proposed by Jaberipur et al. [16] and it is based on the use of PPG scheme [23] and oriented to latency reduction.
- *Gorgin*14. Recently, Gorgin et al. have also introduced several improved and new methods for BCD-digit multiplication [21]. Regarding the most effective in terms of area an delay, they have proposed a new direct truth table approach called New-3 in [21].

Table 1 also contains the FPGA implementation results for the three architectures above. Fig. 8 compares implementation and performance data for the BCD-digit multipliers in Table 1. As it can be seen from this figure, the "Logic&Route Delay" of *DirectTruthTable* and *Gorgin*14 are less than the obtained for our proposals. However, this contribution is oriented to area optimization and $Proposed_1 + Converter_1$ and $Proposed_3 + Converter_1$ architectures occupy less area than the other three designs, getting to reduce the area by 26.32% when compared to *DirectTruthTable*.

To make a further comparison, Table 2 includes the area and delay figures that were summarized by Gorgin et al. in [21] (Table 4 in the cited work). Once more, these delays exclusively refer



(a) Area comparison



(b) Delay comparison

Fig. 9. Comparison of implementation results for BCD-digit multipliers in Table 2.

to the logic and its internal routing, without including those related to input and output buffers and thus excluding any pad delays. These results were obtained for implementations on a Virtex 5 xc5vlx30-3 device. BCD-digit multipliers composed of the proposed binary multipliers ($Proposed_1$, $Proposed_2$ and $Proposed_3$) and $Converter_2$ were also implemented on this Virtex 5 device. As Table 2 shows, the proposed binary multipliers occupy practically the same area as the implementations on Spartan 6 and Virtex 6 devices. Additionally, $Converter_2$ has been selected instead of $Converter_1$ for the implementation of the decimal multipliers on Virtex 5 because this converter offers better area and delay results. It is due to the fact that the synthesis using Virtex 5 is not as effective as the synthesis for more modern FPGA families and high-level descriptions, which is the case for $Converter_1$. Fig. 9 compares implementation and performance data for the BCD-digit multipliers in Table 2. This figure shows that the lowest area is achieved with the proposed BCD multipliers; $Proposed_3 + Converter_2$ occupies 40% less area than $NewGorgin14_3$, which was presented in [21] as the main new contribution. As Fig. 9 also displays, the fastest architecture is $NewGorgin14_3$, but it must be noted that this is a performance-oriented design, while the proposed architectures are area-oriented. In any case, the performance of the presented architectures is faster than most of the rest of compared designs, while they clearly reduce the required area. Thus, the area optimization achieved by our proposals will be more noticeable when $N \times M$-digit BCD multiplications are required. For example, the Standard IEEE 754 for Floating-Point Arithmetic (IEEE 754) [12] requires 15 and 33 decimal digits for decimal64 and decimal128, respectively. Thus, when it is necessary to multiply these data types, besides other logic resources, 15 × 1-digit and 33 × 1-digit BCD multiplications are required. In these cases, the use of the proposed BCD multipliers implies an important area reduction.

6. Conclusions

In this paper, new architectures for BCD-digit multiplication oriented to area reduction have been presented. These proposals are based on the 2-stage BCD multiplication method, which requires a binary product stage and a binary-to-BCD decoding stage. The design of the binary product architecture takes advantage of the non-used symbols in BCD encoding, thus reducing the required area resources. Three different architectures for the binary multipliers were presented and implemented on Xilinx FPGA devices. The evaluation was made by comparing these implementations to the obtained from conventional multipliers and other multipliers described in the literature. Implementation results show that these new multipliers achieve an area reduction up to 38.46%, while the performance is not deteriorated. BCD-digit multipliers composed of these binary multipliers and a binary-to-BCD converter were also implemented on Xilinx devices. The obtained implementations illustrate the benefits of these BCD-digit multipliers regarding area occupancy. The comparison with several existing designs in the literature shows that the proposed ones offer competitive delays and outperform their counterparts in terms of area, as it was intended, getting a 40% area reduction for Virtex 5 implementations. Finally, the presented BCD-digit multiplier, due to its scalability and reduced area, can be employed as the basic IP core for the implementation of decimal arithmetic coprocessor in any IoT device devoted to financial and commercial operations.

Acknowledgment

The authors would like to thank Xilinx Inc. for the licensing of CAD tools and the provision of supporting material.

References

[1] L.K. Wang, M.A. Erle, C. Tsen, E.M. Schwarz, M.J. Schulte, A survey of hardware designs for decimal arithmetic, IBM J. Res. Dev. 54 (2) (2010) 8, http://dx.doi.org/10.1147/JRD.2010.2040930.

[2] A. Nannarelli, Decimal engine for energy-efficient multicore processors, in: 2014 22nd International Conference on Very Large Scale Integration, VLSI-SoC, 2014, pp. 1–6.

[3] N. Dlodlo, M. Mofolo, L. Masoane, S. Mncwabe, G. Sibiya, L. Mboweni, Research Trends in Existing Technologies That Are Building Blocks to the Internet of Things, Springer International Publishing, Cham, 2015, pp. 539–548, http://dx.doi.org/10.1007/978-3-319-06773-5_72.

[4] M. Kavis, The Internet of things will radically change your big data strategy, http://www.forbes.com/sites/mikekavis/2014/06/26/the-internet-of-things-will-radically-change-your-big-data-strategy/, 2015.

[5] J.-L. Sanchez-Romero, H. Mora-Mora, J. Mora-Pascual, A. Jimeno-Morenilla, Function approximation on decimal operands, Digit. Signal Process. 21 (2) (2011) 354–366, http://dx.doi.org/10.1016/j.dsp.2010.06.013, http://www.sciencedirect.com/science/article/pii/S1051200410001557.

[6] L. Han, S.B. Ko, High-speed parallel decimal multiplication with redundant internal encodings, IEEE Trans. Comput. 62 (5) (2013) 956–968, http://dx.doi.org/10.1109/TC.2012.35.

[7] A. Vazquez, E. Antelo, J.D. Bruguera, Fast radix-10 multiplication using redundant bcd codes, IEEE Trans. Comput. 63 (8) (2014) 1902–1914, http://dx.doi.org/10.1109/TC.2014.2315626.

[8] J.K. Toft, A. Nannarelli, Energy efficient FPGA based hardware accelerators for financial applications, in: NORCHIP, 2014, 2014, pp. 1–6.

[9] F. Barranco, M. Tomasi, J. Díaz, M. Vanegas, E. Ros, Pipelined architecture for real-time cost-optimized extraction of visual primitives based on FPGAs, Digit. Signal Process. 23 (2) (2013) 675–688, http://dx.doi.org/10.1016/j.dsp.2012.09.017, http://www.sciencedirect.com/science/article/pii/S1051200412002369.

[10] G. Botella, U. Meyer-Baese, A. García, M. Rodríguez, Quantization analysis and enhancement of a VLSI gradient-based motion estimation architecture, Digit. Signal Process. 22 (6) (2012) 1174–1187, http://dx.doi.org/10.1016/j.dsp.2012.05.013, http://www.sciencedirect.com/science/article/pii/S1051200412001388.

[11] M. George, A.J. B, A. Hussain, S.P. Sreelal, A compact multichannel data acquisition and processing system for IoT applications, in: 2015 International Conference on Advances in Computing, Communications and Informatics, ICACCI, 2015, pp. 1263–1267.

[12] IEEE standard for floating-point arithmetic IEEE Std. 754 (2008) (2008) 1–70, http://dx.doi.org/10.1109/IEEESTD.2008.4610935.

[13] A. Hosseiny, G. Jaberipur, Decimal goldschmidt: a hardware algorithm for radix-10 division, Comput. Electr. Eng. 53 (2016) 40–55, http://dx.doi.org/10.1016/j.compeleceng.2016.06.005, http://www.sciencedirect.com/science/article/pii/S0045790616301604.

[14] J.P. Deschamps, G. Sutter, Decimal division: algorithms and FPGA implementations, in: Programmable Logic Conference (SPL), 2010 VI Southern, 2010, pp. 67–72.

[15] S. Gonzalez-Navarro, C. Tsen, M.J. Schulte, Binary integer decimal-based floating-point multiplication, IEEE Trans. Comput. 62 (7) (2013) 1460–1466, http://dx.doi.org/10.1109/TC.2012.79.

[16] G. Jaberipur, A. Kaivani, Improving the speed of parallel decimal multiplication, IEEE Trans. Comput. 58 (11) (2009) 1539–1552, http://dx.doi.org/10.1109/TC.2009.110.

[17] M. Erle, M. Schulte, Decimal multiplication via carry-save addition, in: Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on, 2003, pp. 348–358.

[18] R. Kenney, M. Schulte, M. Erle, A high-frequency decimal multiplier, in: Computer Design: VLSI in Computers and Processors. ICCD 2004. Proceedings. IEEE International Conference on, 2004, pp. 26–29.

[19] M.A. Erle, E.M. Schwarz, M.J. Schulte, Decimal multiplication with efficient partial product generation, in: 17th IEEE Symposium on Computer Arithmetic, ARITH-17 2005, 2005, pp. 21–28.

[20] A. Vazquez, E. Antelo, P. Montuschi, Improved design of high-performance parallel decimal multipliers, IEEE Trans. Comput. 59 (5) (2010) 679–693, http://dx.doi.org/10.1109/TC.2009.167.

[21] S. Gorgin, G. Jaberipur, R. Hashemi Asl, Efficient asic and FPGA implementation of binary-coded decimal digit multipliers, Circuits Syst. Signal Process. 33 (12) (2014) 3883–3899, http://dx.doi.org/10.1007/s00034-014-9823-4.

[22] M. Fazlali, H. Valikhani, S. Timarchi, H.T. Malazi, Fast architecture for decimal digit multiplication, Microprocess. Microsyst. 39 (4–5) (2015) 296–301.

[23] T. Lang, A. Nannarelli, A radix-10 combinational multiplier, in: Fortieth Asilomar Conference on Signals, Systems and Computers, ACSSC '06, 2006, pp. 313–317.

[24] M.P. Véstias, H.C. Neto, Iterative decimal multiplication using binary arithmetic, in: Programmable Logic (SPL), 2011 VII Southern Conference on, 2011, pp. 257–262.

[25] J. Bhattacharya, A. Gupta, A. Singh, A high performance binary to bcd converter for decimal multiplication, in: 2010 International Symposium on VLSI Design Automation and Test (VLSI-DAT), 2010, pp. 315–318.

[26] O. Al-Khaleel, Z. Al-QudahJ, M. Al-Khaleel, C.A. Papachristou, F. Wolff, Fast and compact binary-to-bcd conversion circuits for decimal multiplication, in: 2011 IEEE 29th International Conference on Computer Design (ICCD), 2011, pp. 226–231.

[27] O. Al-Khaleel, N. Tulic, K. Mhaidat, FPGA implementation of binary coded decimal digit adders and multipliers, in: 8th International Symposium on Mechatronics and Its Applications (ISMA), 2012, 2012, pp. 1–5.

[28] G. Sutter, E. Todorovich, G. Bioul, M. Vazquez, J.-P. Deschamps, FPGA implementations of bcd multipliers, in: International Conference on Reconfigurable Computing and FPGAs, 2009, ReConFig '09, 2009, pp. 36–41.

[29] G. Jaberipur, A. Kaivani, Binary-coded decimal digit multipliers, IEE J. Comput. Digit. Tech. 1 (4) (2007) 377–381, http://dx.doi.org/10.1049/iet-cdt:20060160.

[30] E. Castillo, A. Lloris, A. García, L. Parrilla, D. Morales, Comments on "Fast architecture for decimal digit multiplication", Microprocess. Microsyst. (2016), http://dx.doi.org/10.1016/j.micpro.2016.07.021, http://www.sciencedirect.com/science/article/pii/S0141933116301168.

[31] R. James, T. Shahana, K. Jacob, S. Sasi, Decimal multiplication using compact bcd multiplier, in: International Conference on Electronic Design, ICED 2008, 2008, pp. 1–6.

[32] S. Veeramachaneni, M. Srinivas, Novel high-speed architecture for 32-bit binary coded decimal (bcd) multiplier, in: International Symposium on Communications and Information Technologies, ISCIT 2008, 2008, pp. 543–546.

Encarnación Castillo received the M.A.Sc. degree and Ph.D. degree in electronic engineering from the University of Granada, Spain, in 2002 and 2008, respectively. From 2003 to 2005 she was a research Fellow at the Department of Electronics and Computer Technology at the University of Granada, where she is now an Associate Professor. During her research fellowship, she carried out part of her work at the Department of Electrical and Computer Engineering, Florida State University, Tallahassee. Her research interests include the protection of IP protection of VLSI and FPGAs-based systems, as well as Residue Number System arithmetic, VLSI and FPL signal processing systems and the combination of digital and analog programmable technologies for smart instrumentation for biosignal processing. She has authored more than 40 technical papers in international journals and conferences. She also serves regularly as reviewer for IEEE journals.

Antonio Lloris received the M.Sc. and Ph.D. degrees from the Universidad Complutense, Madrid, Spain. Currently, he is a Full Professor at the

University of Granada, Granada, Spain. He was with the Centro de Investigaciones Técnicas de Guipúzcoa, San Sebastián, Spain, as a Researcher and as a Lecturer with the Escuela Técnica Superior de Ingenieros Industriales de San Sebastián, Guipúzcoa, Spain. He was with the University of Málaga, Málaga, Spain, and the University of Murcia, Murcia, Spain. His research interests include multiple-value logic, testing of digital circuits, and signal processing using the residue number system.

Diego P. Morales received the M.A.Sc. degree in Electronic Engineering and the Ph.D. degree in Electronic Engineering from the University of Granada (Granada, Spain) in 2001 and 2011, respectively. He was an Associate Professor at the Department of Computer Architecture and Electronics, Universidad de Almería (Almería, Spain) before joining the Department of Electronics and Computer Technology at the University of Granada, where he currently serves as an Associated Professor. His current research interests include the combination of digital and analog programmable technologies for smart instrumentation and FPGA-based signal processing systems.

Luis Parrilla received the M.Sc. degree in Physics (majoring in electronics), the M.A.Sc. degree in Electronic Engineering, and the Ph.D. degree in Physics from the University of Granada (Granada, Spain) in 1993, 1995, and 1997, respectively. In 1995 he joined the Department of Electronics and Computer Technology at the University of Granada where he serves as a Professor since 2000. He is author of more than 50 technical papers in international journals and conferences and serves regularly as reviewer for several IEEE and Elsevier journals. His current research interests include the protection of IP cores on VLSI and FPGA based systems, the development of high-performance arithmetic circuits for cryptographic applications, and the design of specific architectures for scientific calculations and biosignal processing over FPLs.

Antonio García received the M.A.Sc. degree in Electronic Engineering (obtaining the Nation Best Academic Record award), in 1995, the M.Sc. degree in Physics (majoring in Electronics), in 1997, and the Ph.D. degree in Electronic Engineering, in 1999, all from the University of Granada. He was an Associate Professor at the Department of Computer Engineering of the Universidad Autónoma de Madrid before joining the Department of Electronics and Computer Technology at the University of Granada, where he actually serves as a Professor. He has authored more than 100 technical papers in international journals and conferences and serves regularly as reviewer for several IEEE and IEE journals. He is also part of the Program Committee for several international conferences on programmable logic. His current research interests include IP protection of VLSI and FPGA-based systems, low-power and high-performance VLSI signal processing systems, and the combination of digital and analog programmable technologies for smart instrumentation for biosignal processing. He is a member of IEEE and a C\&S and SP Society member.

Guillermo Botella received the M.A.Sc. degree in Physics in 1998, the M.Sc. degree in Electronic Engineering, in 2001, and the Ph.D. degree, in 2007, all from the University of Granada (Spain). From 2002 to 2005 he was a research European Fellow at the Department of Architecture and Computer Technology of the Universidad de Granada and the Vision Research Laboratory at University College London. After that he joined as an Associate Professor at the Department of Computer Architecture and Automation of Complutense University of Madrid. He has been visiting professor in 2008–2012 at the Department of Electrical and Computer Engineering, Florida State University, Tallahassee, USA. His current research interests include Digital Signal Processing for Very Large Scale Integration (VLSI), FPGAs, GPGPUs, Vision Algorithms, Cryptography and IP protection of VLSI and FPGA-based systems. He has authored more than 40 papers in international journals and conferences and serves regularly as reviewer for several IEEE journals.