iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 3 **Issue:** V **Month of publication:** May 2015

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Fast Base-10 Multiplication Using XS-3 and ODDS BCD codes

Ch.Balaram Murthy[1], A.Srinivas[2], T.Ragini[3], J.Umesh[4]
[1,2,3,4]*Department of Electronics and communication Engineering,*
[1,2]*Jyothishmathi Institute of Technological Sciences,* [3,4]*Nigama college of Engineering,karimnagar*

Abstract—We present the algorithm and architecture of a BCD parallel multiplier that exploits some properties of two different redundant BCD codes to speed up its computation: the redundant BCD excess-3 code (XS-3), and the overloaded BCD representation (ODDS). In addition, new techniques are developed to reduce significantly the latency and area of previous representative high performance implementations. Partial products are generated in parallel using a signed-digit base-10 recoding of the BCD multiplier with the digit set [-5,5], and a set of positive multiplicand multiples (0X, 1X, 2X, 3X, 4X, 5X) coded in XS-3.

Keywords— Parallel multiplication, decimal hardware, overloaded BCD representation, redundant excess-3 code, redundant arithmetic

I. INTRODUCTION

DECIMAL fixed-point and floating-point formats are important in financial, commercial, and user-oriented computing, where conversion and rounding errors that are inherent to floating-point binary representations cannot be tolerated [3]. The new IEEE 754-2008 Standard for Floating Point Arithmetic which contains a format and specification for decimal floating-point (DFP) arithmetic [1], has encouraged a significant amount of research in decimal hardware [6] Since area and power dissipation are critical design factors in state-of-the-art DFPUs, multiplication and division are performed iteratively by means of digit-by-digit algorithms [4], [5], and therefore they present low performance.

Moreover, the aggressive cycle time of these processors puts an additional constraint on the use of parallel techniques [6], for reducing the latency of DFP multiplication in high-performance DFPUs. Thus, efficient algorithms for accelerating DFP multiplication should result in regular VLSI layouts that allow an aggressive pipelining. Hardware implementations normally use BCD instead of binary to manipulate decimal fixed-point operands and integer significant of DFP numbers for easy conversion between machine and user representations BCD encodes a number X in decimal (non-redundant base-10) format, with each decimal digit Xi € [0, 9] represented in a 4-bit binary number system. However, BCD is less efficient for encoding integers than binary, since codes 10 to 15 are unused. BCD carry-save and signed-digit base-10 arithmetic offer improvements in performance with respect to non redundant BCD. However, the resultant VLSI implementations in current technologies of multi operand adder trees may result in more irregular layouts than binary carry-save adders (CSA) and compressor trees. The overloaded BCD (or ODDS—overloaded decimal digit set) representation was proposed to improve decimal multi operand addition and sequential and parallel [12], [13] decimal multiplications. In this code, each 4-bit binary value represents a redundant base-10 digit Xi € [0, 15]. The ODDS presents interesting properties for a fast and efficient hardware implementation of decimal arithmetic:(1) it is a redundant decimal representation so that it allows carry-free generation of both simple and complex decimal multiples (2X, 3X, 4X, 5X, 6X,. . .) and addition,(2) since digits are represented in the binary number system, digit operations can be performed with binary arithmetic, and (3) unlike BCD, there is no need to implement additional hardware to correct invalid 4-bit combinations. A disadvantage with respect to signed-digit and self-complementing codes, is a slightly more complex implementation of 9's complement operation for negation of operands and subtraction We propose the use of a general redundant BCD arithmetic (that includes the ODDS, XS-3 and BCD representations) to accelerate parallel BCD multiplication in two ways:

Partial product generation (PPG). By generating positive multiplicand multiples coded in XS-3 in a carry free form. An advantage of the XS-3 representation over non-redundant decimal codes (BCD and 4221/5211 [3]) is that all the interesting multiples for decimal partial product generation, including the 3X multiple, can be implemented in constant time with an equivalent delay of about three XOR gate levels. Moreover, since XS-3 is a self-complementing code, the 9's complement of a positive multiple can be obtained by just inverting its bits as in binary.

Partial product reduction (PPR). By performing the reduction of partial products coded in ODDS via binary carry-save arithmetic.

Partial products can be recoded from the XS-3 representation to the ODDS representation by just adding a constant factor into the partial product reduction tree. The resultant partial product reduction tree is implemented using regular structures of binary carry-save adders or compressors. The 4-bit binary encoding of ODDS operands allows a more efficient mapping of decimal algorithms into binary techniques. By contrast, signed-digit base-10 and BCD carry-save redundant representations require specific base-10 digit adders

II. REDUNDANT BCD REPRESENTATIONS

The proposed decimal multiplier uses internally a redundant BCD arithmetic to speed up and simplify the implementation. This arithmetic deals with base-10 ten's complement integers of the form:

$$Z= -S_z \times 10^d + \sum_{i=0}^{d-1} 1 \times Z_i \times 10^d \quad (1)$$

where d is the number of digits, S_z is the sign bit, and $Z_i \in [\,l\text{-}e, m\text{-}e]$ is the i^{th} digit $0 \le l \le e$; $9 + e \le m \le 2^4 - 1 (= 15)$. Parameter e is the excess of the representation and usually takes values 0 (non excess), 3 or 6. The redundancy index ρ is defined as $\rho = m - l + 1 - r$ [12], being $r = 10$.

The value of Z_i depends on the decimal representation parameterized by (l, m, e). We use a 4-bit encoding to represent digits Z_i. This allows us to manage decimal operands in different representations with the same arithmetic, such as BCD ($Z_i \in [0, 9]$; e = 0, l = 0,m = 9, $\rho = 0$), BCD excess-3 ($Z_i \in [0, 9]$, e = 3, l = 3,m = 12, $\rho = 0$), BCD excess-6 ($Z_i \in [0, 9]$, e = 6, l = 6, m = 15; $\rho = 0$), and redundant representations ($\rho > 0$), such as the ODDS representation ($Z_i \in [0, 15]$, e = 0, l = 0, m = 15, $\rho = 6$), or the XS-3 representation ($Z_i \in [-3,12]$, e = 3, l = 0,m =15, $\rho = 6$).

On the other hand, the binary value of the 4-bit vector representation of Z_i is given by

$$[Z_i] = \sum_{j=0}^{3} 1 \times Z_{i,j} \times 2^j , \quad (2)$$

$Z_{i,j}$ being the j^{th} bit of the i^{th} digit. Therefore, the value of digit Zi can be obtained by subtracting the excess e of the representation from the binary value of its 4-bit encoding, that is, $Z_i = [Zi] - e$

This binary encoding simplifies the hardware implementation of decimal arithmetic units, since we can make use of state-of-the-art binary logic and binary arithmetic techniques to implement digit operations. In particular, the ODDS representation presents interesting properties (redundancy and binary encoding of its digit set) for a fast and efficient implementation of multi operand addition. Moreover, conversions from BCD to the ODDS representation are straightforward, since the digit set of BCD is a subset of the ODDS representation.

We use a SD base-10 recoding of the BCD multiplier [3], which requires to compute a set of decimal multiples ({-5X, . . . , 0X, . . . , 5X}) of the BCD multiplicand. The main issue is to perform the x3 multiple without long carry-propagation. For input digits of the multiplicand in conventional BCD (i.e., in the range [0, 9], e = 0, $\rho = 0$), the multiplication by 3 leads to a maximum decimal carry to the next position of 2 and to a maximum value of the interim digit (the result digit before adding the carry from the lower position) of 9. Therefore the resultant maximum digit (after adding the decimal carry and the interim digit) is 11. Thus, the range of the digits after the x3 multiplication is in the range [0, 11]. Therefore the redundant BCD representations can host the resultant digits with just one decimal carry propagation.

An important issue for this representation is the ten's complement operation. Since after the recoding of the multiplier digits, negative multiplication digits may result, it is necessary to negate (ten's complement) the multiplicand to obtain the negative partial products. This operation is usually done by computing the nine's complement of the multiplicand and adding a one in the proper place on the digit array.

The nine's complement of a positive decimal operand is given by

$$-10^d + \sum_{i=0}^{d-1} 1 \times (9 - Z_i) \times 10^i \quad (3)$$

The implementation of $(9 - Z_i)$ leads to a complex implementation, since the Z_i digits of the multiples generated may take values higher than 9. A simple implementation is obtained by observing that the excess-3 of the nine's complement of an operand is equal to the bit complement of the operand coded in excess-3.

TABLE 1
Nine's Complement for the XS-3 Representation

Digit			Nine's Complement		
4-bit Encoding	Z_i	$[Z_i]$	4-bit Encoding	$9 - Z_i$	$[9 - Z_i]$ $(=15 - [Z_i])$
0000	-3	0	1111	12	15
0001	-2	1	1110	11	14
0010	-1	2	1101	10	13
0011	0	3	1100	9	12
0100	1	4	1011	8	11
0101	2	5	1010	7	10
0110	3	6	1001	6	9
0111	4	7	1000	5	8
1000	5	8	0111	4	7
1001	6	9	0110	3	6
1010	7	10	0101	2	5
1011	8	11	0100	1	4
1100	9	12	0011	0	3
1101	10	13	0010	-1	2
1110	11	14	0001	-2	1
1111	12	15	0000	-3	0

In Table 1 we show how the nine's complement can be performed by simply inverting the bits of a digit Z_i coded in XS-3. At the decimal digit level, this is due to the fact that:

$$(9 - Z_i) + 3 = 15 - (Z_i + 3) \quad (4)$$

for the ranges $Z_i \in [-3, 12]$ ($Z_i \in [0, 15]$). Therefore to have a simple negation for partial product generation we produce the decimal multiples in an excess-3 code. The negation is performed by simple bit inversion, that corresponds to the excess-3 of the nine's complement of the multiple. Moreover, to simplify the implementation we combine the multiple generation stage and the digit increment by 3(to produce the excess-3) into a single module by using the XS-3 code

In summary, the main reasons for using the redundant XS-3 code are: (1) to avoid long carry-propagations in the generation of decimal positive multiplicand multiples, (2) to obtain the negative multiples from the corresponding positive ones easily, (3) simple conversion of the partial products generated in XS-3 to the ODDS representation for efficient partial product reduction

III. HIGH-LEVEL ARCHITECTURE

The high-level block diagram of the proposed parallel architecture for dxd-digit BCD decimal integer and fixed-point multiplication is shown in Fig. 1. This architecture accepts conventional (non-redundant) BCD inputs X, Y , generates redundant BCD partial products PP, and computes the BCD product P =X * Y . It consists of the following three stages1: (1) parallel generation of partial products coded in XS-3, including generation of multiplicand multiples and recoding of the multiplier operand, (2) recoding of partial products from XS-3 to the ODDS representation and subsequent reduction, and (3) final conversion to a non-redundant 2d-digit BCD product.

Volume 3 Issue V, May 2015
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

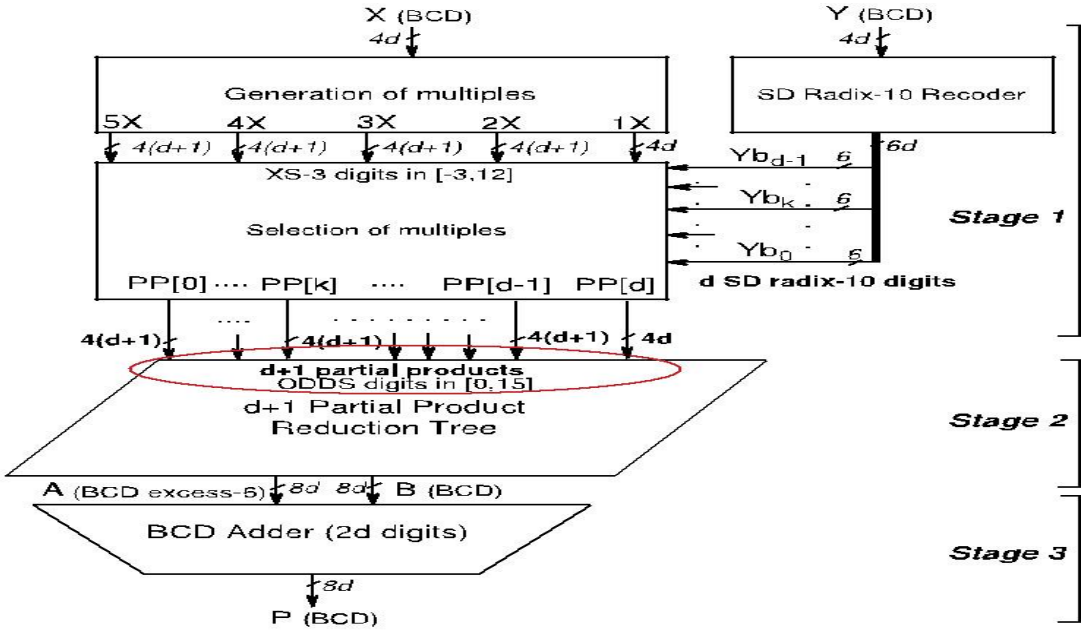


Fig.1. Combinational SD Base -10 architecture.

Stage 1) Decimal partial product generation. A SD base-10 recoding of the BCD multiplier has been used. This recoding produces a reduced number of partial products that leads to a significant reduction in the overall multiplier area Therefore, the recoding of the d-digit multiplier Y into SD base-10 digits Yb_{d-1}; . . . ; Yb_0, produces d partial products PP[d – 1], . . . ,PP[0], one per digit; note that each Yb_k recoded digit is represented in a 6–bit hot-one Code to be used as control input of the multiplexers for selecting the proper multiplicand multiple, {-5X, . . . ,-1X, 0X, 1X, . . . 5X}. An additional partial product PP[d] is produced by the most significant multiplier digit after the recoding, so that the total number of partial Products generated is d + 1.

Stage 2) Decimal partial product reduction. In this stage, the array of d + 1 ODDS partial products are reduced to two 2d-digit words (A, B). Our proposal relies on a binary carry save adder tree to perform carry-free additions of the decimal partial products. The array of d + 1 ODDS partial products can be viewed as adjacent digit columns of height $h \leq d + 1$. Since ODDS digits are encoded in binary, the rules for binary arithmetic apply within the digit bounds, and only carries generated between base-10 digits (4-bit columns) contribute to the decimal correction of the binary sum. That is, if a carry out is produced as a result of a 4-bit (modulo 16) binary addition; the binary sum must be incremented by 6 at the appropriate position to obtain the correct decimal sum (modulo 10 additions).

Stage 3) Conversion to (non-redundant) BCD. We consider the use of a BCD carry-propagate adder [9] to perform the final conversion to a non-redundant BCD product P = A+ B. The proposed architecture is a 2d-digit hybrid parallel prefix/carry-select adder, the BCD Quaternary Tree adder The sum of input digits A_i , B_i at each position i has to be in the range [0,18] so that at most one decimal carry is propagated to the next position i + 1 [2].Furthermore, to generate the correct decimal carry, the BCD addition algorithm implemented requires $A_i + B_i$ to be obtained in excess-6. Several choices are possible. We opt for representing operand A in BCD excess-6 ($A_i \in [0, 9]$, $[A_i] = A_i + e$, e = 6), and B coded in BCD ($B_i \in [0, 9]$, e = 0).

IV. DECIMAL PARTIAL PRODUCT GENERATION

The partial product generation stage comprises the recoding of the multiplier to a SD base-10 representation, the calculation of the multiplicand multiples in XS-3 code and the generation of the ODDS partial products. The SD base-10 encoding produces d SD base-10 digits $Yb_k \in [-5, 5]$, with k = 0, . . . ,d -1, Y_{d-1} being the most significant digit (MSD) of the multiplier [9]. Each digit Yb_k is represented with a 5-bit hot-one code ($Y1_k$, $Y2_k$, $Y3_k$, $Y4_k$, $Y5_k$) to select the appropriate multiple {1X, . . . ,5X} with a 5:1 mux and a sign bit Ys_k that controls the negation of the selected multiple. The negative multiples are obtained by ten's complementing the positive ones. This is equivalent to taking the nine's complement of the positive multiple and then adding1. The nine's complement

Volume 3 Issue V, May 2015
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

can be obtained simply by bit inversion.

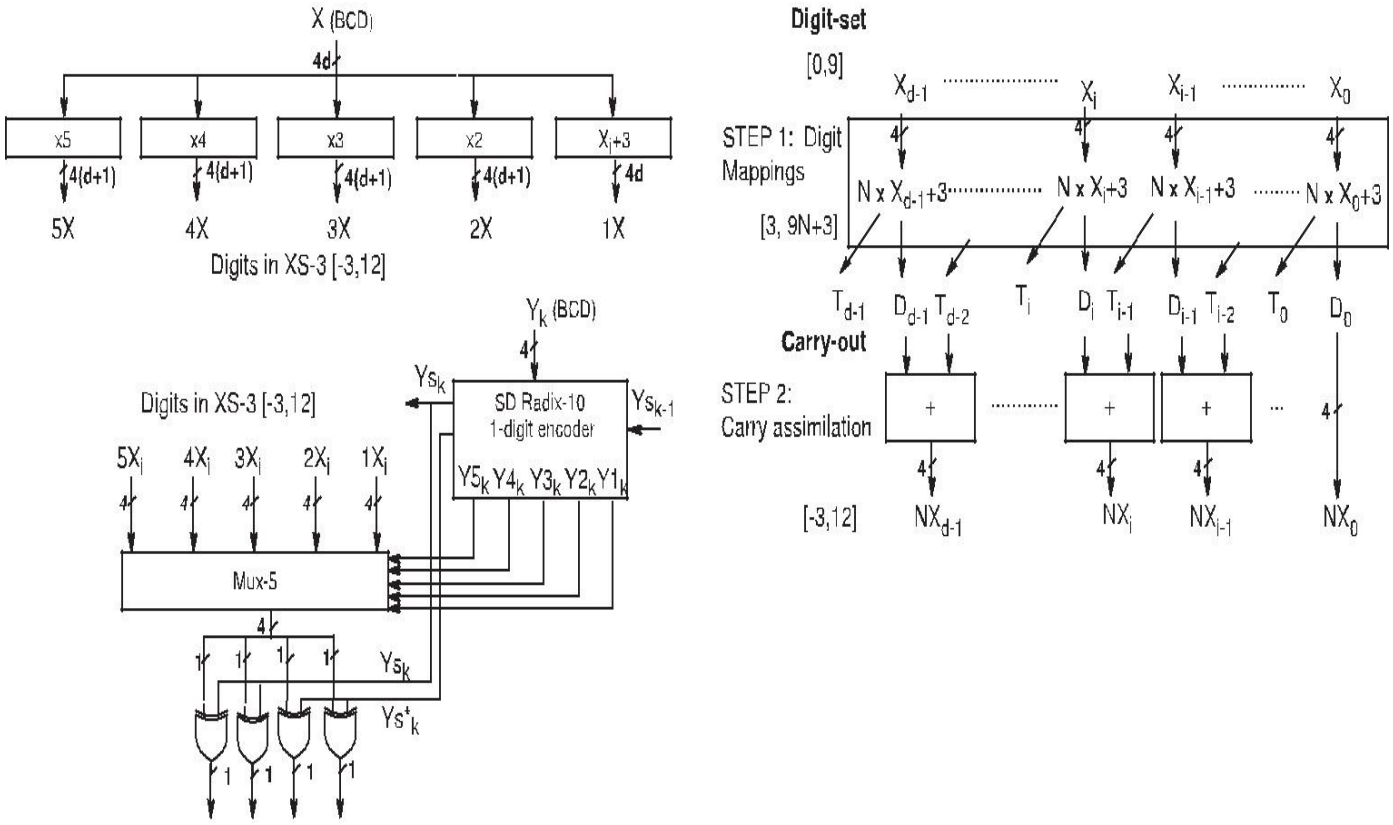


Fig. 2. SD base-10 generation of a partial product digit.

This needs the positive multiplicand multiples to be coded in XS-3, with digits in [-3, 12].The d least significant partial products PP [d – 1], . . . ,PP [0] are generated from digits Yb_k by using a set of 5:1 muxes, as shown in Fig. 2. The xor gates at the output of the mux invert the multiplicand multiple, to obtain its 9's complement, if the SD base-10 digit is negative (Ys_k =1).

A. Generation of the Multiplicand Multiples

We denote by NX ∈ {1X, 2X, 3X, 4X, 5X}, the set of multiplicand multiples coded in the XS-3 representation, with digits NX_i ∈ [-3, 12], being $[NX_i] = NX_i + 3 \in [0, 15]$ the corresponding value of the 4-bit binary encoding of NX_i given by Equation (2).

Fig. 2 shows the high-level block diagram of the multiples generation with just one carry propagation. This is performed in two steps:

1) digit recoding of the BCD multiplicand digits X_i into a decimal carry $0 \leq Ti \leq T_{max}$ and a digit $-3 \leq D_i \leq 12 - T_{max}$, such as
$$D_i + 10 \times T_i = (N \times X_i + 3; \quad (5)$$
being T_{max} the maximum possible value for the decimal carry.

2) The decimal carries transferred between adjacent digits are assimilated obtaining the correct 4-bit representation of XS-3 digits NX_i, that is

$$[NX_i] = Di + T_{i-1}\ ; \quad [NX_i] \in [0, 15] \quad (NX_i \in [-3,12]) \quad (6)$$

B. Most-Significant Digit Encoding

The MSD of each PP[k], $PP_d[k]$, is directly obtained in the ODDS representation. Note that these digits store the carries generated in the computation of the multiplicand multiples
and the sign bit of the partial product. For positive partial products we have

Volume 3 Issue V, May 2015
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

$$PP_d[k] = T_{d-1} \quad (7)$$

with $T_{d-1} \in \{0,1,2,3,4\}$ (see Table 2). For negative partial products, the ten's complement operation leads to

$$PP_d[k] = -10 + (9 - T_{d-1}) = -1 - T_{d-1} \quad (8)$$

with $T_{d-1} \in \{0,1,2,3,4\}$. Therefore the two cases can be expressed as

$$PP_d[k] = -Ys_k + (-1)^{Ys_k} T_{d-1} \quad (9)$$

Since we need to encode $PP_d[k]$ in the ODDS range [0, 15], we add and subtract 8 in Eq. (9), resulting in

$$[PP_d[k]] = -8 + [PP_d[k]] \quad (10)$$

with

$$[PP_d[k]] = 8 - Ys_k + (-1)^{Ys_k} T_{d-1}$$

Note that the term $[PP_d[k]]$ is always positive. Specifically, for positive partial products ($Ys_k = 0$), this term results in $8 + T_{d-1}$ that is within the range [8], [12] (since $0 \le T_{d-1} \le 4$). For negative partial products ($Ys_k = 1$), this term results in $7 - T_{d-1}$, that is within the range [3], [7]. All of the -8 terms of the different partial products are grouped together in a constant $-8 \times \sum_{k=0}^{d-1} 1 \times 10^{k+d}$ that is added as a constant correction term to the results of the reduction array.

Therefore, the $PP_d[k]$'s are encoded as $[PP_d[k]]$ without the -8 terms, which are added later , with only positive values of the form

$$[PP_d[k]] = \begin{cases} (8 + T_{d-1}) & , \text{ if}(Ys_k = 0); \\ (7 - T_{d-1}) & , \text{ if}(Ys_k = 1); \end{cases} \quad (11)$$

resulting in $[PP_d[k]] \in [3, 12]$

This encoding is implemented at bit level as an inversion of the 3 LSB's of T_{d-1} if $Ys_k = 1$ and the concatenation of the MSB Ysk^- bar.

TABLE 2
Preferred Digit Recoding Mappings for NX Multiples

	1X			2X			3X			4X			5X		
X_i	X_i+3	T_i	D_i	$(X_i \times 2)+3$	T_i	D_i	$(X_i \times 3)+3$	T_i	D_i	$(X_i \times 4)+3$	T_i	D_i	$(X_i \times 5)+3$	T_i	D_i
0	3	0	3	3	0	3	3	0	3	3	0	3	3	0	3
1	4	0	4	5	0	5	6	0	6	7	0	7	8	0	8
2	5	0	5	7	0	7	9	0	9	11	1	1	13	1	3
3	6	0	6	9	0	9	12	0	12	15	1	5	18	1	8
4	7	0	7	11	1	1	15	1	5	19	1	9	23	2	3
5	8	0	8	13	1	3	18	1	8	23	2	3	28	2	8
6	9	0	9	15	1	5	21	2	1	27	2	7	33	3	3
7	10	0	10	17	1	7	24	2	4	31	2	11	38	3	8
8	11	0	11	19	1	9	27	2	7	35	3	5	43	4	3
9	12	0	12	21	1	11	30	2	10	39	3	9	48	4	8

V. EVALUATION AND COMPARISON

The proposed combinational architectures for BCD 16x16-digit and 34 x 34-digit multipliers are evaluated and compared with other representative BCD multipliers. The area and delay figures of our architectures were obtained from an area-delay evaluation model

for static CMOS gates, and validated with the synthesis of verified RTL models coded in VHDL.

A. Evaluation

As stated above, the evaluation has been performed in two steps. First, a technological independent evaluation using a model for static CMOS circuits based on Logical Effort (LE) has been carried out, and then the results obtained with this model have been validated with the synthesis and functional verification of the RTL model.

B. Comparison

Table 3 shows the area and delay estimations obtained from synthesis for some representative BCD sequential and combinational multipliers. As far as we know, the most representative High-performance BCD multipliers are 16 x16-digit combinational [12], and sequential [9],[10], implementations.

The area and delay figures shown in Table 3 correspond to the minimum delay point of each implementation, and were obtained from the synthesis results provided in the respective reference, except for the two multipliers of reference [12], which correspond to an Estimation obtained by their authors using a LE-based model. The comparison ratios are given with respect to the area and delay figures of a 53-bit binary Booth base-4 multiplier

The PPG of multipliers [7], is based on a SD base-5 scheme, that generates 32 BCD partial products for a 16-digit multiplier. Though it only requires simple constant time delay BCD multiplicand multiples, the 9's complement operation for obtaining the negative multiples is more complex than a simple bit inversion. The partial product reduction implement is a BCD carry-save adder tree build of BCD digit adders. On the other hand, the BCD partial products are reduced in [7] by using counters that compute the binary sum of each column of digits sum, and subsequent binary to decimal conversions [7]. The BCD multiplier pre-computes all the positive decimal multiplicand multiples {0X. . . 9X}. The delay of PPG is reduced by representing the complex operands (3X, 6X, 7X, 8X, 9X) as the sum of two simpler multiples. The number of partial products generated is therefore equivalent to that of the SD base-5 scheme. The PPR tree is implemented with BCD digit adders This has the disadvantage of a large area compared to the other BCD multipliers analysed.

The two 16 x16-digit BCD multipliers of [12] implement an easy-multiple PPG [9] (only pre-computes {2X, 4X, 5X) that produces 32 BCD partial products. The intermediate decimal partial product sums are computed in overloaded BCD to speed up the PPR evaluation. The delay-improved design uses a tree structure built of five levels overloaded BCD digit adders, while the area-improved design replaces two levels of these custom designed adders by three levels of 4: 2 compressors and a binary counter. This reduces the area consumption but at the cost of introducing a significant latency penalty.

TABLE 3

Synthesis Results for Fixed-Point Multipliers

Architecture	Delay (# FO4)	Latency # Cycles	Latency (# FO4)	Latency Ratio	Throughput Mult./Cycle	Area (NAND2)	Area Ratio
Combinational multipliers:							
Binary 53 × 53-bit:							
Booth radix-4	31.1	1	31.1	1.00	1	34000	1.00
BCD 16 × 16-digit:							
Ref. [19]	58.9	1	58.9	1.90	1	68000	2.00
Ref. [7]	55.8	1	55.8	1.80	1	68000	2.00
Ref. [16]	53.5	1	53.5	1.70	1	79600	2.35
Delay-Improved [12]	47	1	47	1.50	1	48600	1.45
Area-Improved [12]	51	1	51	1.65	1	39100	1.15
SD Radix-10 [30]	48	1	48	1.55	1	44500	1.30
SD Radix-5 [30]	46.5	1	46.5	1.50	1	49700	1.45
Ref. [14]	43.1	1	43.1	1.40	1	49900	1.45
Proposed	41.5	1	41.5	1.35	1	44200	1.30
BCD 34 × 34-digit:							
Proposed	56	1	56	1.80	1	120600	3.55
Sequential multipliers (BCD 16 × 16-digit):							
Ref. [10]	16	20	320	10.20	1/17	16000	0.50
Ref. [9]	14.7	20	294	9.45	1/17	18550	0.55
Ref. [17]	12.7	24	305	9.80	1/17	31500	0.90

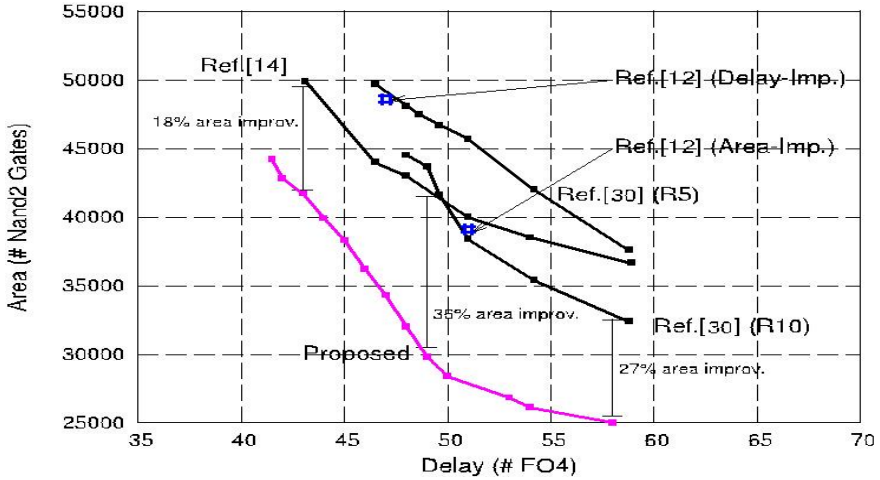


Fig. 3. Area-delay space for the fastest 16x16-digit mults.

To compare the high hardware cost of a combinational Decimal128 implementation, we also include in Table 3 the area and delay figures obtained for our 34 x34-digit BCD multiplier. Due to the tight area and power consumption constraints of current DFUs [5], a sequential architecture seems a more realistic solution than a fully pipelined implementation
for a commercial Decimal128 multiplier.

Finally, we present a more detailed comparison of the fastest BCD 16 x16-digit combinational multipliers (SD Base-5 and SD Base-10 [12], and the proposed one) in terms of latency and area. The corresponding area delay synthesis values are shown in Fig. 3 We have directly introduced in the figure the area-delay curves of referenced multipliers [9] and [11] as provided by their authors, since all of them were synthesized using 90 nm CMOS standard cell libraries and similar conditions. The area-delay points for the two multipliers of reference [12] correspond to an estimation obtained by their authors using a LE-based model.

From the area-delay space represented in Fig. 10, we observe that our proposed decimal multiplier has an area improvement roughly in the range 20-35 percent depending on the target delay. On the other hand, for the minimum delay point (44FO4), the proposed multiplier is still competitive with the fastest design shown in

More recently, the authors of reference [12] have presented in a comparison study between their delay improved multiplier and the multiplier of reference based on synthesis results using a TSMC 130 nm standard CMOS process under typical conditions (1.2 V, 25 C). They show that for the minimum delay point of each one of the two area-delay curves obtained, the delay-improved multiplier [12] is 20 percent faster and has 10 percent less area compared to the design of Therefore, according to the curve corresponding to the design presented should be to the left of the area-delay points corresponding to the delay-improved design presented in [12].

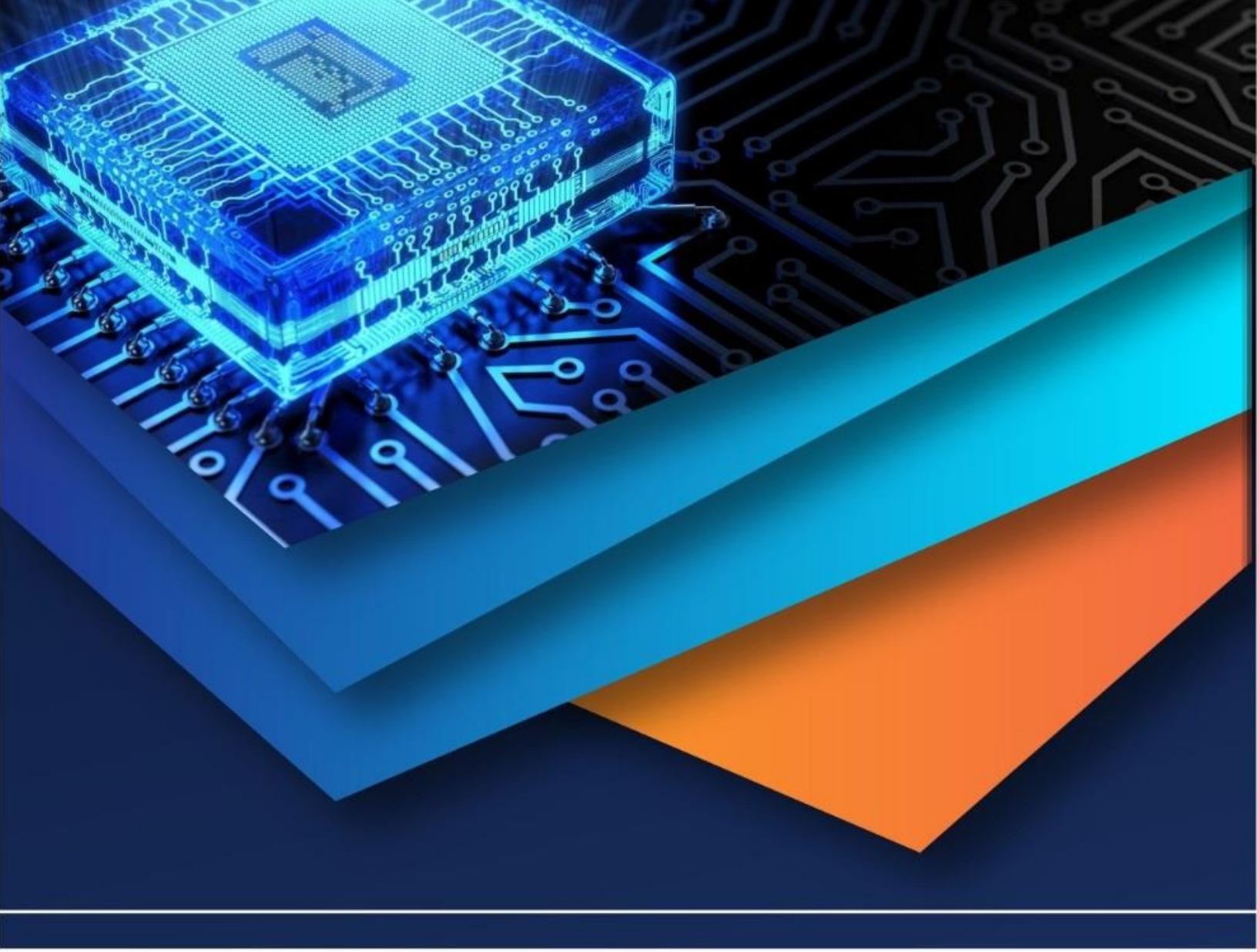
VI. CONCLUSION

In this paper we have presented the algorithm and architecture of a new BCD parallel multiplier. The improvements of the proposed architecture rely on the use of certain redundant BCD codes, the XS-3 and ODDS representations. Partial products can be generated very fast in the XS-3 representation using the SD base-10 PPG scheme: positive multiplicand multiples (0X, 1X, 2X, 3X, 4X, 5X) are pre-computed in a carry-free way, while negative multiples are obtained by bit inversion of the positive ones. On the other hand, recoding of XS-3 partial products to the ODDS representation is straightforward. The ODDS representation uses the redundant digit-set [0, 15] and a 4-bit binary encoding (BCD encoding), which allows the use of a binary carry-save adder tree to perform partial product reduction in a very efficient way. The area and delay figures estimated from both a theoretical model and synthesis show that our BCD multiplier presents 20-35 percent less area than other design

REFERENCES

[1] A. Aswal, M. G. Perumal, and G. N. S. Prasanna, “On basic financial decimal operations on binary machines,” IEEE Trans. Comput., vol. 61, no. 8, pp. 1084–1096, Aug. 2012.
[2] M. F. Cowlishaw, E. M. Schwarz, R. M. Smith, and C. F. Webb, “A decimal floating-point specification,” in Proc. 15th IEEE Symp. Comput. Arithmetic, Jun.

2001, pp. 147–154.
[3] M. F. Cowlishaw, “Decimal floating-point: Algorism for computers,” in Proc. 16th IEEE Symp. Comput. Arithmetic, Jul. 2003, pp. 104–111.
[4] S. Carlough and E. Schwarz, “Power6 decimal divide,” in Proc. 18th IEEE Symp. Appl.- Specific Syst., Arch., Process., Jul. 2007, pp. 128–133.
[5] S. Carlough, S. Mueller, A. Collura, and M. Kroener, “The IBM zEnterprise-196 decimal floating point accelerator,” in Proc. 20th IEEE Symp. Comput. Arithmetic, Jul. 2011, pp. 139–146.
[6] L. Dadda, “Multioperand parallel decimal adder: A mixed binary and BCD approach,” IEEE Trans. Comput., vol. 56, no. 10, pp. 1320–1328, Oct. 2007.
[7] L. Dadda and A. Nannarelli, “A variant of a Base-10 combinational multiplier,” in Proc. IEEE Int. Symp. Circuits Syst., May 2008,pp. 3 370–3373.
[8] L. Eisen, J. W. Ward, H.-W. Tast, N. Mading, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough, “IBM POWER6 accelerators: VMX and DFU,” IBM J. Res. Dev., vol. 51, no. 6, pp. 663–684, Nov. 2007.
[9] M. A. Erle and M. J. Schulte, “Decimal multiplication via carrysave addition,” in Proc. IEEE Int. Conf Appl.-Specific Syst., Arch., Process., Jun. 2003, pp. 348–358.
[10] M. A. Erle, E. M. Schwarz, and M. J. Schulte, “Decimal multiplication with efficient partial product generation,” in Proc. 17th IEEE Symp. Comput. Arithmetic, Jun. 2005, pp. 21–28.
[11] Faraday Tech. Corp. (2004). 90nm UMC L90 standard performance low-K library (RVT). [Online]. Available: http://freelibrary.faraday- tech.com/
[12] S. Gorgin and G. Jaberipur, “A fully redundant decimal adder and its application in parallel decimal multipliers,” Microelectron. J., vol. 40, no. 10, pp. 1471–1481, Oct. 2009.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY