International Journal of Emerging Engineering Research and Technology
Volume 3, Issue 9, September, 2015, PP 68-76
ISSN 2349-4395 (Print) & ISSN 2349-4409 (Online)

Design of BCD Parrell Multiplier Using Redundant BCD Codes

M.Saritha[1], B.Laxman[2], Dr.B.R.Vikram[3]

[1]*Department of ECE, Vijay Rural Engineering College, Nizamabad, India (PG Scholar)*
[2]*Department of ECE, Vijay Rural Engineering College, Nizamabad, India (Associate Professor)*
[3]*Department of ECE, Vijay Rural Engineering College, Nizamabad, India (Principal)*

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 radix-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. This encoding has several advantages. First, it is a self-complementing code, so that a negative multiplicand multiple can be obtained by just inverting the bits of the corresponding positive one. Also, the available redundancy allows a fast and simple generation of multiplicand multiples in a carry free way. Finally, the partial products can be recoded to the ODDS representation by just adding a constant factor into the partial product reduction tree. Since the ODDS uses a similar 4-bit binary encoding as non-redundant BCD, conventional binary VLSI circuit techniques, such as binary carry-save adders and compressor trees, can be adapted efficiently to perform decimal operations. To show the advantages of our architecture, we have synthesized a RTL model for 16-digit and 34-digit multiplications and performed a comparative survey of the previous most representative designs.

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

INTRODUCTION

The common multiplication method is “add and shift” algorithm. In parallel multipliers number of partial products to be added is the main parameter that determines the performance of the multiplier. To improve the speed of the BCD multiplier, radix-10 algorithm is one of the most popular algorithms. To achieve speed improvements one digit BCD adder Tree can be used to reduce the number of sequential adding stages. Further by combining both one digit BCD adder Tree and radix-10 technique we can see advantages in one multiplier. However with increasing parallelism, the amount of shifts between the partial products and intermediate sums to be added will increase which may result in reduced delay, increase in silicon area due to irregularity of structure and also increased power consumption due to increase in interconnect resulting from complex routing. On the other hand “serial-parallel“ multipliers compromise speed to achieve better performance for area and power consumption. The selection of a parallel or serial multiplier actually depends on the nature of application. In this lecture we introduce the multiplication algorithms and architecture and compare them in terms of speed, area, power and combination of these metrics. Hardware implementations normally use BCD instead of binary to manipulate decimal fixed-point operands and integer significands of DFP numbers for easy conversion between machine and user representations. BCD encodes a numberXin decimal (non-redundant radix-10) format, with each decimal digit Xi 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. Moreover, the implementation of BCD arithmetic has

Address for correspondence:

merugu.saritha@gmail.com

more complications than binary, which lead to area and delay penalties in the resulting arithmetic units. A variety of redundant decimal formats and arithmetic have been proposed to improve the performance of BCD multiplication. The BCD carry-save format represents a radix-10 operand using a BCD digit and a carry bit at each decimal position. It is intended for carry-free accumulation of BCD partial products using rows of BCD digit adders arranged in linear or tree-like configurations. Decimal signed-digit (SD) representations to allow decimal carry-free addition. BCD carry-save and signed-digit radix-10 arithmetic offer improvements in performance with respect to nonredundant BCD. However, the resultant VLSI implementations in current technologies of multioperand adder trees may result in more irregular layouts than binary carry-save adders (CSA) and compressor trees

HIGH LEVEL ARCHITECTURE

The high-level block diagram of the proposed parallel architecture for d×d 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=X*Y. It consists of the following three stages:

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.

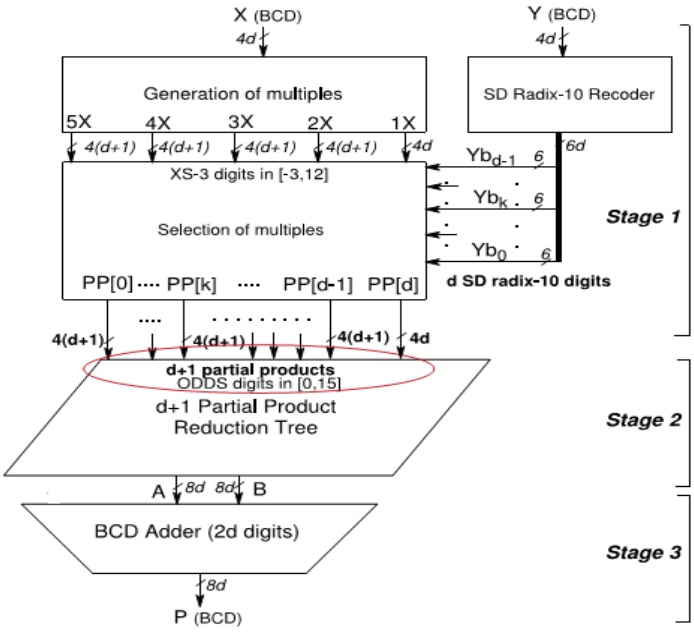


Fig1.

Stage 1) Decimal partial product generation: A SDradix-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 radix-10 digits Ybd-1;...; Yb0, produces d partial products PP[d-1];...;PP[0], one per digit ;note that each Ybk 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. In contrast to our previous SD radix-10 implementations, 3X is obtained in a reduced constant time delay (~3 XOR-gate delays) by using the XS-3 representation. Moreover, a negative multiple is generated from the correspondent positive one

by a bitwise XOR operation. Consequently, the latency is reduced and the hardware implementation is simplified. The scheme proposed in also produces 3Xin constant time but using redundant signed-digit BCD arithmetic.

Stage 2) Decimal partial product reduction. In this stage, the array of d+1ODDS 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≤d+1. Since ODDS digits are encoded in binary, the rules for binary arithmetic apply within the digit bounds, and only carries generated between radix-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 addition). Two previous designs implement tree structures for the addition of ODDS operands. In the non speculative BCD adder, a combinational logic block is used to determine the sum correction after all the operands have been added in a binary CSA tree, with the maximum number of inputs limited to 19 BCD operands[2.] By contrast, in our method the sum correction is evaluated concurrently with the binary carry-save additions using columns of binary counters. Basically we count the number of carries per decimal column and then a multiplication by 6 is performed (a correction by 6 for each carry-out from each column). The result is added as a correction term to the output of the binary carry-save reduction tree. This improves significantly the latency of the partial product reduction tree. Moreover, the proposed architecture accepts an arbitrary number of ODDS or BCD operand inputs. Some of PPR tree structures presented in (the area-improved PPR tree) also exploit a similar idea, but rely on a custom designed ODDS adder to perform some of the stage reductions. Our proposal aims to provide an optimal reuse of any binary CSA tree for multi operand decimal addition, as it was done for the 4221 and 5211 decimal coding.

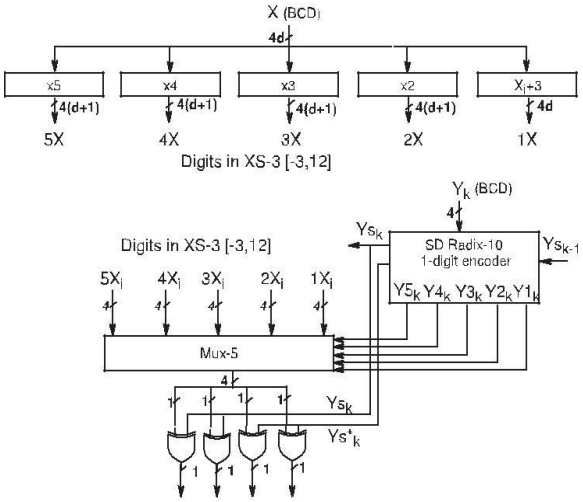
Stage 3) Conversion to (non-redundant) BCD. We consider the use of a BCD carry-propagate adder to perform the final conversion to a non-redundant BCD product P=[A+B]. The proposed architecture is a 2d-digit hybrid parallel prefix/carry-select adder, the BCD Quaternary Tree adder (see Section 6). The sum of input digits Ai , Bi 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. Furthermore, to generate the correct decimal carry, the BCD addition algorithm implemented requires Ai + Bi to be obtained in excess-6. Several choices are possible. We opt for representing operand A in BCD excess-6 (Ai belongs to [0;9], [Ai]=Ai + e, e=6), and B coded in BCD (Bi belongs to[0;9], e=0).

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

DECIMAL PARTIAL PRODUCT GENERATION

The partial product generation stage comprises the recoding of the multiplier to a SD radix-10 representation, the calculation of the multiplicand multiples in XS-3 code and the generation of the ODDS partial products.The SD radix-10 encoding produces d SD radix-10 digits Yb_k € [-5; 5], with k = 0; . . . ; d-1, Y_{d-1} being the most significant digit (MSD) of the multiplier. 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 adding 1. As we have shown in Section 2, the nine's complement can be obtained simply by bit inversion. This needs the positive multiplicand multiples to be coded in XS-3, with digits in [-3; 12].



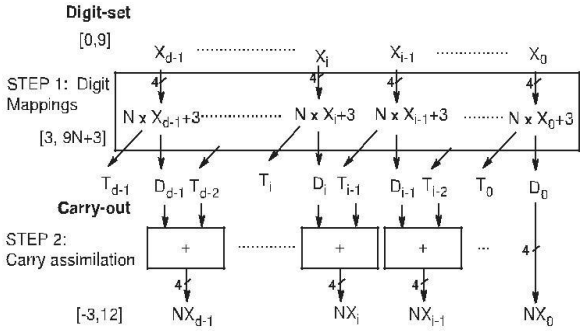


Fig2. *SD radix-10 generation of a partial product digit.* **Fig3.** *Generation of a decimal multiples NX*

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 radix-10 digit is negative (Ys_k = 1).

On the other hand, if the signals ($Y1_k$; $Y2_k$; $Y3_k$; $Y4_k$; $Y5_k$) are all zero then PP [k] = 0, but it has to be coded in XS-3 (bit encoding 0011). Then, to set the two least significant bits to 1, the input to the XOR gate is $Ys_k^* = Ys_k \vee Yb_k$ is zero (V denotes the boolean OR operator), where Yb_k is zero equals 1 if all the signals ($Y1_k$; $Y2_k$; $Y3_k$; $Y4_k$; $Y5_k$) are zero.

In addition, the partial product signs are encoded into their MSDs . The generation of the most significant partial product PP [d] is described in Section 3.2, and only depends on Ys_{d-1}, the sign of the most significant SD radix-10 digit.

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 ϵ [-3; 12], being $[NX_i] = NX_i + 3$ € [0; 15] the corresponding value of the 4-bit binary encoding of NX_i given by Equation $[Z_i] = \sum_{j=0}^{3} z_{i,j} \times 2^j z_{i,j}$ being the jth bit of ith digit

Fig. 3 shows the high-level block diagram of the multiples generation with just one carry propagation. This is per-formed in two steps:

1) digit recoding of the BCD multiplicand digits X_i into a decimal carry $0 <= T_i <= T_{max}$ and a digit $-3 <= D_i <= 12 - T_{max}$, such as

$D_i + 10 \times T_i = (N \times X_i) + 3;$ (1)

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 The constraint for NX_i still allows different implementations for NX. For a specific implementation, the mappings for T_i and D_i have to be selected. Table 2 shows the preferred digit recoding for the multiples NX.

Then, by inverting the bits of the representation of NX, operation defined at the ith digit by

$$\overline{NX_i} = 15 - [NX_i];$$

we obtain $\overline{NX}$. Replacing the relation between NX_i and $[NX_i]$ in the previous expression, it $\overline{NX_i} = 15 - (NX_i + 3) = (9 - NX_i) + 3$:

That is, $\overline{NX}$ is the 9's complement of NX coded in XS-3, with digits $\overline{NXi} \in [-3; 12]$ and

$[\overline{NX_i}] = \overline{NX_i} + 3 \in [0, 15]$.

	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

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

$PP_d[k] = T_{d-1}$ (2)

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

$PP_d[k] = -10 + (9 - T_{d-1}) = -1 - T_{d-1}$ (3)

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

$PP_d[k] = -Ys_k + (-1)^{Ysk} T_{d-1}$ (4)

Since we need to encode $PP_d[k]$ in the ODDS range [0; 15], we add and subtract 8 in Eq. (4), resulting in

$PP_d[k] = -8 + [PP_d[k]];$ (5)

with

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

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 $0 \leq T_{d-1} \leq 4$). For negative partial

products (Ys_k = 1), this term results in 7 - T_{d-1}, that is within the range. All of the -8 terms of the different partial products are grouped together in a constant -8 $\times\sum_{k=0}^{d-1} 10^{k+d}$ that is added as a constant correction term k=0 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 (see Section 4.3), with only positive values of the form resulting in $[PP_d[k]] \in [3; 12]$.

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

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

Correction Term

The resultant partial product sum has to be corrected off-the-critical-path by adding a pre-computed term, $f_c(d)$, which only depends on the format precision d. This term has to gather:

a) The -8 constants that have not been included in the MSD encoding and

b) A -3 constant for every XS-3 partial product digit (introduced to simplify the nine's complement operation).

Actually, the addition of these -3 constants is equivalent to convert the XS-3 digits of the partial products to the ODDS representation. Note that the 4-bit encoding of a XS-3 digit NX_i (or 9 - NX_i) represents an ODDS digit with value $[NX_i] = NX_i + 3 \in [0; 15]$ (or $[9 - NX_i] = 15 - [NX_i] \in [0; 15]$).

The pre-computed correction term is given by

$$f_c(d) = -8 \times \sum_{k=0}^{d-1} 10^{k+d} - 3 \times \left(\sum_{i=0}^{d-1} (i+1)10^i + \sum_{i=0}^{d-2} (d-1-i)10^{i+d} \right).$$

Particularizing for d = 16 and d = 34 digit operands, the following expressions for the correction term in 10's complement are obtained:

$f_c(16) = -10^{32}$ + 0740740740740741703703703703703 7

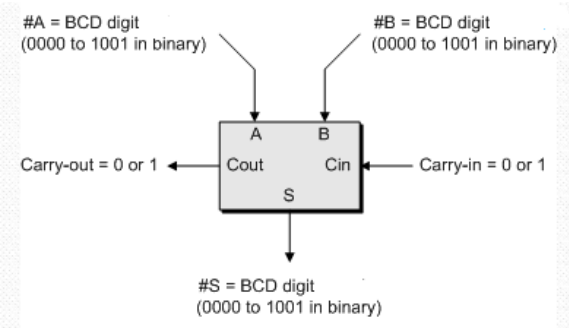
$f_{c(}34) = -10^{68}$ + 074074074074…… 07417037037037:

The correction term is allocated into the array of d + 1 partial products coded in ODDS (digits in [0; 15]), as we show in the next section.

Proposed Decimal Partial Product Reduction

The concepts underlying BCD (binary-coded-decimal) representations. In particular, we considered unsigned versus 10s-complement versions of BCD numbers. In this column we are going to consider how we go about adding and subtracting unsigned BCD values .Just to remind ourselves, if we are using an 8-bit byte to represent two "unsigned" BCD digits, then #00 to #99 in BCD equates to 0 to +99 in decimal (we will use "#" characters to indicate BCD values).So how would we go about adding two such bytes together? One technique would be to create a special adder that can directly add two BCD digits together, along with a C_{in} ("carry-in") bit, and generate a single BCD digit as output along

with a C_{out} ("carry-out") bit, as shown in



Remembering that each BCD digit can only support values of 0000 to 1001 in binary (0 to 9 in decimal), this means a result greater than 9 will cause a carry-out. For example, 3 + 4 = 7 in decimal, so if we present BCD digits of #3 and #4 to our adder, we expect to see a result of #7 with C_{out} = 0. By comparison, 6 + 8 = 14 in decimal, so if we present BCD digits of #6 and #8 to our adder, we expect to see a result of #7 with C_{out} = 1.

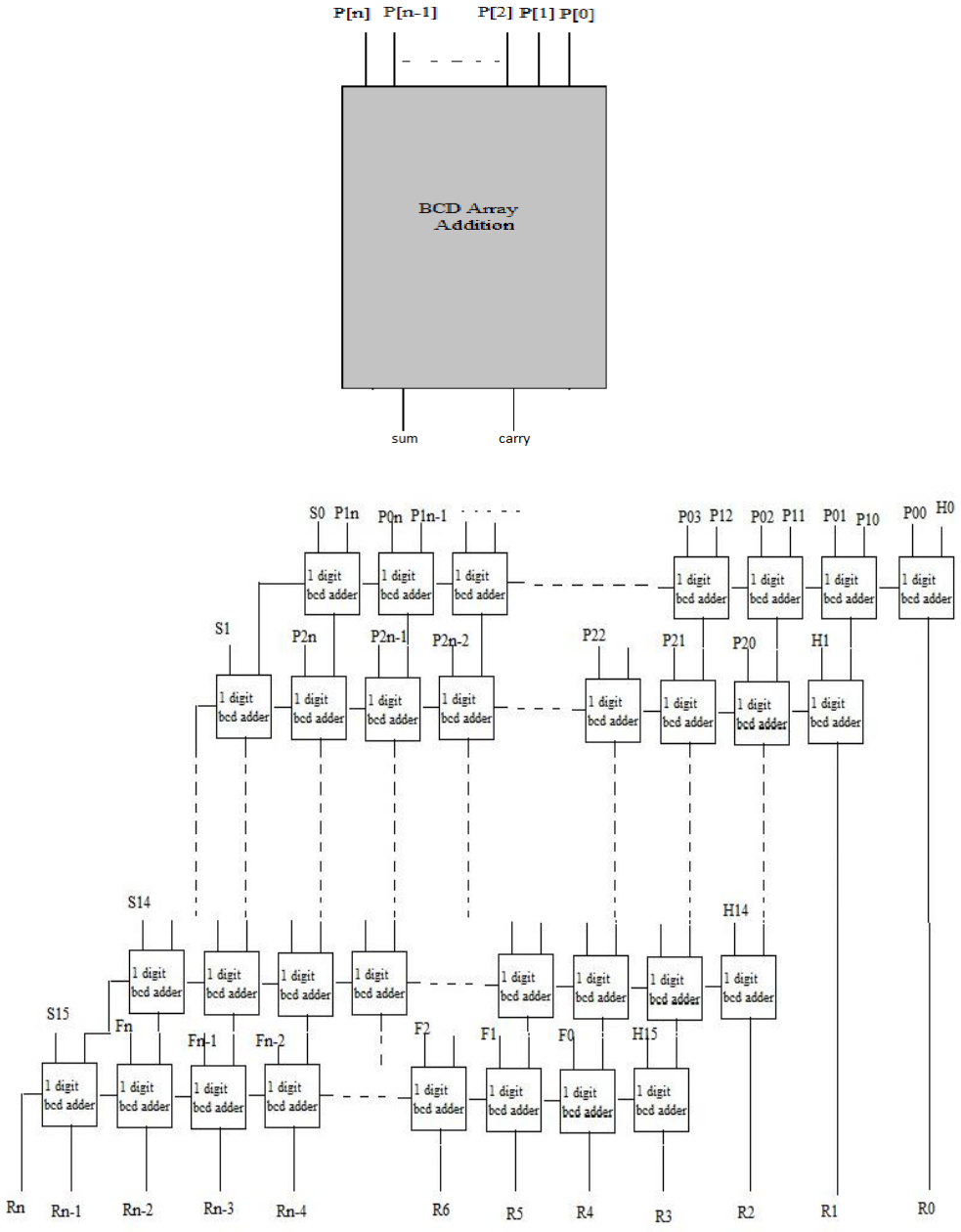
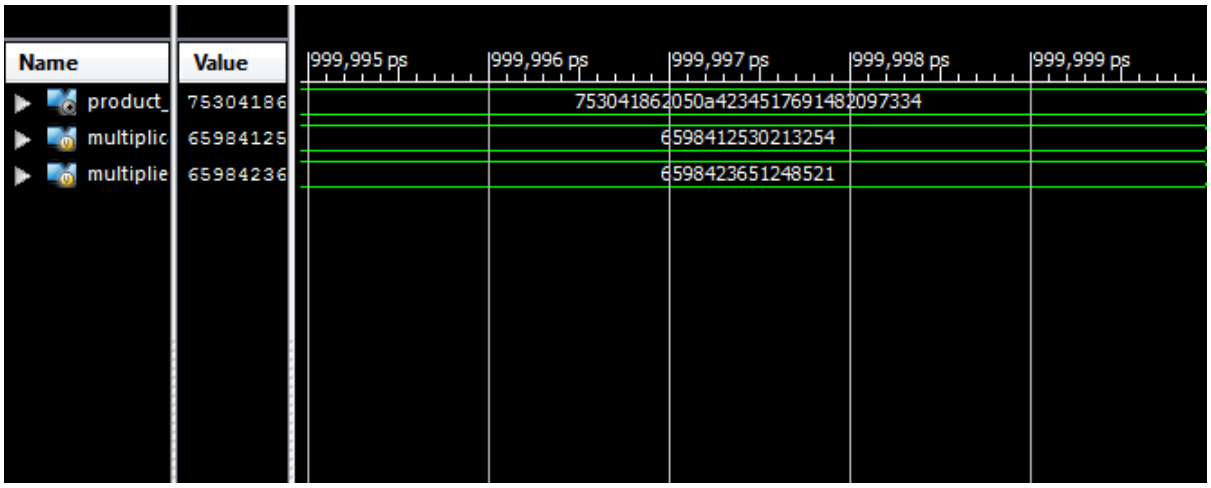


Fig3

SIMULATION RESULTS



CONCLUSION

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 radix-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.

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, "Adecimal floating-point specification," inProc. 15th IEEE Symp.Comput. Arithmetic, Jun. 2001, pp. 147–154.

[3] M. F. Cowlishaw, "Decimal floating-point: Algorism for computers," inProc. 16th IEEE Symp. Comput. Arithmetic, Jul. 2003,pp. 104–111.Fig. 10. Area-delay space for the fastest 16-digit mults.V AZQUEZ ET AL.: FAST RADIX-10 MULTIPLICATION USING REDUNDANT BCD CODES 1913

[4] S. Carlough and E. Schwarz, "Power6 decimal divide," inProc. 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," inProc. 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 Radix-10 combinational multiplier," inProc. IEEE Int. Symp. Circuits Syst., May 2008,pp. 3370–3373.

[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," inProc. 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.

AUTHOR'S BIOGRAPHY

M.SARITHA. pursued Btech (ece)(2009) from Kshatriya College of Engineering, Armoor, Nizamabad. She is currently pursuing Mtech from VIJAY RURAL ENGINEERING COLLEGE, Nizamabad.

B.LAXMAN pursued Mtech from NIIT WARANGAL in 2009 currently working as associate professor in ECE department of VIJAY RURAL ENGINEERING COLLEGE since 2009, he has 6 years of teaching experience.

Dr.B.R.VIKRAM received Doctor of Philosophy in Electronics & Communication Engineering (Digital Image Processing) from Bundelkhand University, Uttar Pradesh, in the Year 2008 Pursued Master of Engineering Instrumentation from Swami Ramanand Teerth Marathwada University, Nanded, in the year 2004 and Bachelor of Engineering in Electronics & Instrumentation from Pune University, Shahada, in the year 1995. Worked As Senior Engineer for Diplomatic Engineering LLC, Dubai (U.A.E) for Dubai Rail Transit System in the Year 2005