International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

FPGA Implementation of New Architecture for a 16-Bit High Speed Redundant BCD Coded Radix-10 Multiplier

K.Suresh[1] : J.E.N.Abhilash [2] &Ch.K.L.Rao[3]
[1]M.Tech Student Department of ECE,SCET, Narasapur, Andhra Pradesh. INDIA
[2] M.Tech(Ph.D)Associative Professor Department of ECE,SCET, Narasapur, Andhra Pradesh. INDIA
[3]Assistant Professor Department of ECE,SCET, Narasapur, Andhra Pradesh. INDIA
sureshvarma.0447@gmail.com[1], abhilash.jen@gmail.com[2], klr_mani786@yahoo.co.in[3]

ABSTRACT:
The paper mainly focus on a development of a new architecture of a BCD parallel multiplier that utilizes 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 this we have developed some new techniques to reduce the latency and area of some previously high-performance implementations. Here the main role plays by the Partial product generation in parallel with the signed-digit radix-10 recoding of the BCD multiplier with the digit set of [-5, 5], and a set of positive multiplicand multiples such as (0X, 1X, 2X, 3X, 4X, 5X) coded in XS-3.By using the XS-3 approach with signed digit radix-10 recoding had several advantages like mostly it is self inverting code, which means negative number can be obtained by just inverting the bits of a positive number. Also, its available redundancy allows a simple and fast multiplicand multiples with a carry-free way and lastly, the partial products can be recoded to ODDS representation by simply adding a constant factor to its partial product reduction tree. Since the ODDS uses a 4-bit binary encoding similarly as non redundant BCD, conservative binary VLSI circuit technique. We developed a new approach of BCD addition for the final stage. The above developed architecture of has been synthesized a RTL model and given better performance compared to old version multipliers

Keywords: Binary Coded Decimal (BCD); Overloaded BCD (ODDS); signed-digit; Radix-10; Carry Save Adder.

1.INTRODUCTION

DECIMAL fixed-point and floating-point formats are very important in commercial, financial, and user-oriented computing, where the conversion and rounding errors are inherent tothe floating-point binary representations cannot be tolerated [3]. The new IEEE 754-2008 Standard for Floating- Point Arithmetic, which contain the design and specification of a decimal floating-point (DFP) arithmetic [1], [2],has been encouraged a significant amount of research in decimal hardware [6], [9],

[10], [28],. Moreover, at present IBM Power and organization families of microprocessors [5], [8], [23], and the Fujitsu Sparc X microprocessor [26] are oriented to servers and mainframes and they already include fully IEEE 754-2008 compliant decimal floating-point units (DFPUs) for Decimal64 (16 precision digits) and Decimal128 (34 precision digits) formats. Since the area and power dissipation factors are critical design in state-of-the-art DFPUs, division and multiplication are performed constantly by means of digit-by-digit algorithms [4],

[5], and therefore they present a low performance. However, the processor aggressive cycle time puts in an additional constraint with the use of parallel techniques [6], [19], [30] for reducing latency of the DFP multiplication in an high speed performance

DFPUs. Therefore, for the most capable algorithms accelerating DFP multiplication should result in regular VLSI layouts that may allows an aggressive pipelining.

For an easy conversion between machine and user representations [21], [25], hardware iplementations normally used a BCD rather than binary to manipulate decimal fixed-point operands and integer significands of DFP numbers. BCD encodes a number X in decimal format (non-redundant radix-10), with each decimal digit Xi ; represents a 4-bit binary number system. However, Binary is more efficient for encoding integers than BCD, since the codes 10 to 15 are unused in BCD. Moreover, the functioningof arithmetic BCD increases more difficulties than binary, which direct to increase area and delay in the resulting arithmetic units. A variety of techniques have been proposed to increase the performance of BCD multiplication, such as redundant decimal formats and arithmetics. The BCD carry-save format [9] represents a radix-10 operand using a BCD digit and a carry bit at each decimal position. It is intended to generate a carry-free accumulation of BCD partial products using rows of BCD digit adders arranged in linear [9], or tree-like structured configuration's [19]. Decimal signed-digit representation [10],[14], rely on a redundant digit set $\{-a, \ldots, 0, \ldots, a\}$, $5 \leq a \leq 9$, to allow decimal carry-free addition.

Moreover, these codes are self-complementing, so the 9's complement of a digit, necessary for reversal, it can be easily obtained bybit-inversion of 4-bit representation.A disadvantage of 4221 and 5211 codes, is the use of a non-redundant radix-10 digit-set [0, 9]equivalent as BCD. Hence, the redundancy is constrained to the digit bounds, suchthat complex decimal multiples ofX, cannot be obtained in a carry-freeway.

In this work, we mainly focus on the improvement of parallel decimal multiplication by utilizng the redundancy of two decimal representations: the ODDS and the redundant BCD excess-3 (XS-3) representation, a self-complementing code with the digit set [3, 12].For the recording of the BCD multiplier digits we use a redundant digit set, the signed-digit radix-10 recoding [30], that is, the recodedsign digits are in the set $\{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$. For this digit set, the main setback is to carry out the multiple without a long carry-propagation (note that, they are easy multiples for decimal [30] and that is generated in two consecutive operations). We propose the advantage of general redundant BCD arithmetic which includes the ODDS, For the particular digit set, the main problem is to generate the multiple without a long carry-propagation (note that they are easy multiples of decimal [30] and that is generated as two consecutive operations). We introduce the use of general redundant BCD arithmetic (which includes the ODDS,XS-3 and BCD representations) to accelerate the parallel BCD multiplication can be done in two ways.Partial product generation (PPG) is used to generate positive multiplicand multiples coded in XS-3 in a carry-free form. An advantage of XS-3 representation over an non-redundant decimal codes (BCD and 4221/5211 [30]) is that all multiples for decimal partial product generation, with the X multiple, can be implemented in constant time with an corresponding delay of about three XOR gate levels. Furthermore, since XS-3 is a self complementing code, The 9's complement of positive multiples can be generated by just complementing its bits as in binary form. Partial product reduction (PPR) is uesd to performing the reduction of partial products coded in ODDS by binary carry-save arithmetic. By just adding a constant factor to the partial product reduction tree, the partial products can be easily recoded from XS-3 to the ODDS representation. Hence the resultant partial product reduction tree can be implemented usingastandard 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. Hence the signed-digit radix-10 and BCD carry-save redundant representations requires specific radix-10 digit adders [14], [22], [27].

The paper is structured as follows. Section 2 introduces theredundant BCD representation is used in this work. Section 3 intoduces the high level algorithm and architecture of the proposed BCD parallel multiplier. In Section 4 we introduce the techniques for the generation of decimal partial products. Finally the conversion to a non-redundant BCD product and Decimal partial product reduction are detailed in Sections 5 and 6 respectively.

2. REDUNDANT BCD REPRESENTATIONS

The proposed decimal multiplier uses internally a disused BCD arithmetic to speed up its computation and simplify the implementation. This arithmetic deals with a radix-10 ten's complementintegers is of theform: $Z = -s_z \times 10^d + \sum_{i=0}^{d-1} Z_i \times 10^i,$ where d represents the number of digits, s_z represents the sign bit, On the other hand, the binary value of a 4-bit vector representation of Zi is given by

$$[Z_i] = \sum_{j=0}^{3} z_{i,j} \times 2^j,$$

$z_{i;j}$ is the jth bit of ith digit. Hence, the value of the digit Zi can be done by decreasing the excess e from the representation of the of its 4-bit binary encoding, that is,

$$Z_i = [Z_i] - e.$$

Note that, the bit-weighted code such as BCD and ODDS representation uses the 4-bit binary encoding (or BCD encoding) defined in an Expression (2). Hence the, ZiZi for operands of Z represent in BCD or ODDS. This binary encoding of the hardware implementation simplifies the decimal arithmetic units, hence we can employ the state-of-art binary logic and binary arithmetic techniques are to implement the digit operations. In general, the ODDS representation presenta very interesting properties, such as redundancy and binary encoding of its digit set, for an high efficiency and fast implementation of multi-operand addition. Furthermore, the conversions from a BCD to ODDS representation is a straight- forward, hence the digit set of a BCD is subset of the ODDS representation. In our paper we make use of SD radix-10 recoding of the BCD multiplier [30], requires to compute a set of decimal multiples $(\{-5X, \ldots, 0X, \ldots, 5X\})$ of the BCD multiplicand. Here the main issue is to perform the X3 multiple without a long carry-propagation.

Forthe input digits of a multiplicand in an conventional BCD (i.e., the range [0, 9], e , r), the multiplication by 3 which leads to a maximum decimal carry to 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. Hence the resultant maximum digit (after adding the decimal carry and the interim digit) is 11. Thus, the range of the digits after the 3 multiplication is in the range [0, 11]. Hence the redundant BCD representations can represent the resultant digits with justa one decimal carry propagation. Here the important issue for this representation is ten's complement operation. Since after recoding the multiplier digits, negative multiplication digits may come into extent, it is necessary to negate (ten's complement) the multiplicand to obtain the negative partial products. This operation can be done by computing the nine's complement of a multiplicand and by adding a one in the proper place of the digit array. The nine's complement of the positive decimal operand cn be obtained by

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

The implementation of 9-Zi which leads to a very complex implementation, hence the digitsZi of the multiples can be generated may take the values higher than 9. A simple implementation is obtained by just observing the excess-3 of the nine's complement of an operand which is equal to the bit-complement of an operand coded in excess-3.

3. HIGH-LEVEL ARCHITECTURE

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

Stage 1:- Decimal partial product generation. A SDradix-10 of the recoding BCD multiplier has been used. This recoding reduces a number of partial products which leads to a significant reduction of the overall multiplier area [29]. Hence, the recoding of the d-digit multiplier Y into SD radix-10 digits Y ; ;Yb ,produces d partial products PP d ; ;PP ,one per digit; note that each

Ybk recoded digit can be represented in a 6–bit hot-one code and this code can be used as control input of the multiplexers to select a proper multiplicand multiple, An additional partial product PPd is produced by the most significantmultiplier digit after the recoding, so that the total number of partial products generated is d.

Stage 2:- Decimal partial product reduction. In this stage the array of d ODDS partial products can be reduced to two d-digit words (A and B). This proposal relies on a binary carry- save adder tree to perform a carry-free additions of decimal partial products. The array of d ODDS partial products can be observed as adjacent digit columns of height h<=d . Hence the ODDS digits are to be encoded in binary, the rules of a binary arithmetic can be applied within the digit bounds, hence only carries are generated between radix-10 digits (4-bit columns) contribute a decimal correction of the binary sum. That is, if a carry out is generated as a result of 4-bit (modulo 16) binary addition, the binary sum is incremented by 6 at the certain position to obtain a correct decimal sum (modulo 10 addition).

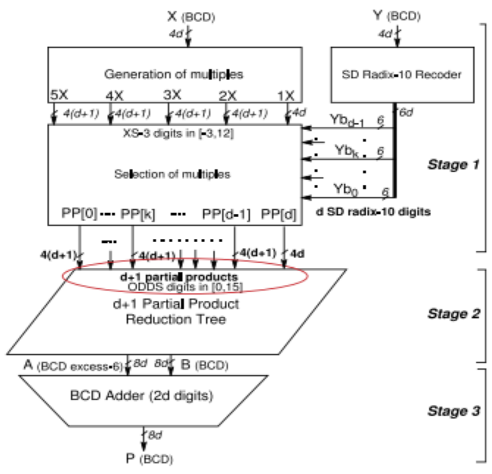


Fig. 1. Combinational SD radix-10 architecture.

Stage 2:- Decimal partial productreduction. In this stage the array of d ODDS partial productscan bereduced to two d-digit words (A and B). This proposal relies on a binary carry- save adder tree to performa carry-free additions of decimal partial products. The array of d ODDS partial products can be observed as adjacent digit columns of height h<=d . Hence the ODDS digits are to beencoded in binary, the rules of a binary arithmetic can be applied within the digit bounds, hence only carries are generated between radix-10 digits (4-bit columns) contribute a decimal correction of the binary sum. That is, if a carry out is generated as a result of 4-bit (modulo 16) binary addition, the binary sum is incremented by 6 at the certain position to obtain a correct decimal sum (modulo 10 addition).

Two early designs [12], [18] implement a tree structures for an addition of ODDS operands. In the non speculative BCD adder [18], a combinational logic block can be used to determine the sum correction once all the operands can be added in a binary CSA tree, with the maximum number of inputs limited to 19 BCD operands. In our method the sum correction can be evaluated concurrently with binary carry-save additions using the columns of binary counters. Normally we count the number of carries per decimal column, since a multiplication by 6 is performed (a correction by 6 for each carry-out from each column). The result is added to the correction term to the output of a binary carry-save reduction tree. Significantly this can improves the latency of the partial product reduction tree. Furthermore, the proposed architecture can accepts the arbitrary number of ODDS or input operands of the BCD. Some of PPR tree structures presented in [12] (the area-improved PPR tree)alsoutilize a similar idea, but they rely on a custom designed ODDS adder to perform the reductions of stages. Our proposal aims is to provide an optimal reuse of any binary CSA tree for themultioperand decimal addition, as it can done in [31] for the 5211 and 4221 decimal codings.

Stage 3:- Conversion to (non redundant) BCD. We have to consider the use of BCD carry-propagate adder [29] to perform the final conversion to an non-redundant BCD product P= A+ B. Here the proposed architecture is a BCD Quaternary Tree adder, a d-digit hybrid parallel prefix/carry-select adder,. 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 [22]. Moreover, to generate a correct decimal carry, the BCD addition algorithm implemented requires Ai+ Bi must be obtained in excess-6. Several choices are possible. We opt for representing operand A in BCD excess-6

4.DECIMAL PARTIAL PRODUCT GENERATION

Thedecimal partial product generation stage comprises of recoding the multiplier to a SD radix-10 representation, the generation of the ODDS partial products and the calculation of multiplicand multiples in XS-3 code.

The negative multiples can be easily obtained by ten's complementing of the positive ones. This is equivalent to the nine's complement of the positive multiple and then adding 1. We have been shown in Section 2, the nine's complement can be obtained easily by bit inversion. This needs the positive multiplicand multiples to be coded in XS-3 with digits in range (-3-12).The d least significant partial products PP[d]....PP[0] are generated from digits Ybk by using a set of 5:1 muxes, as shown in Fig_2. The xor gates at the output of the mux is to invert the multiplicand multiples, to obtain the 9's complement, if the SD radix-10 digit is negative (Ysk =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 can aslo 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 YskYskYbk is zero (denotes the boolean OR operator), where Ybkiszero 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 (seen in Section 4.2). The generation of a most significant partial product PP(d),it only depends on Ysd , the sign of most significant SD radix-10 digit.

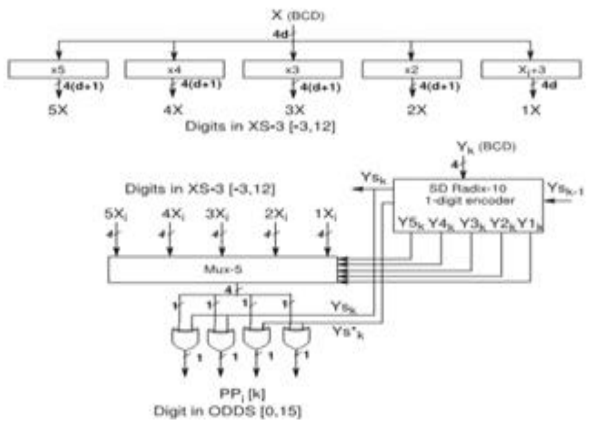


Fig. 2. SD radix-10 generation of a partial product digit.

4.1 Generation of the Multiplicand Multiples

We denote by NX multiples(1X, 2X, 3X, 4X, 5X) are the set of multiplicand multiples which are coded in the XS-3 representation, with the digits NXi in the range(-3 to 12) , being [NXi]=NXi+3 with in the range(0 to 15), the corresponding value of a 4-bit binary encoding of NXi given by Equation (2).Fig. 3 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 Xi into a decimal carry $0 \leq T_i \leq T_{max}$ and a digit $-3 \leq D_i \leq 12 - T_{max}$, such as $D_i + 10 \times T_i = (N \times X_i) + 3$, being Tmax the maximum possible value for the decimal carry.

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

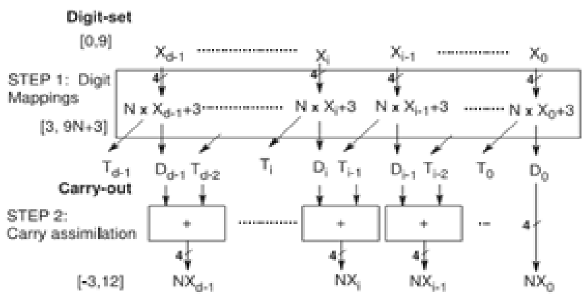


Fig. 3. Generation of a decimal multiples NX.

$$[NX_i] = D_i + T_{i-1}, [NX_i] \in [0,15](NX_i \in [-3,12]).$$

$$[NX_i] = D_i + T_{i-1}, [NX_i] \in [0,15](NX_i \in [-3,12]). \qquad (6)$$

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 follows that

$$\overline{NX_i} = 15 - (NX_i + 3) = (9 - NX_i) + 3.$$

That is, $\overline{NX}$ is the 9's complement of NX coded in XS-3, with digits $\overline{NX_i} \in [-3,12]$ and $[\overline{NX_i}] = \overline{NX_i} + 3 \in [0,15]$.

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

Most-Significant Digit Encoding

The most significant digit(MSD) of each PP[k] , PPd[k] , is directly obtained in ODDS representation. Hence these digits can store the carries that are to be generated in the computation of the multiplicand multiples and the sign bit of the partial product. For the positive and negative partial products we have

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

4.3.Correction Term:

The resultant partial product sum must be corrected off the-critical-path by adding a pre-computed term, $f_c$which only depends on format precision d. This term has to gather: (a) the constants that does not included in the MSD encoding and (b) a constant for an every XS-3 partial product digit (introduced to simplify the nine's complement operation). Actually, the sum of these constants are 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.

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

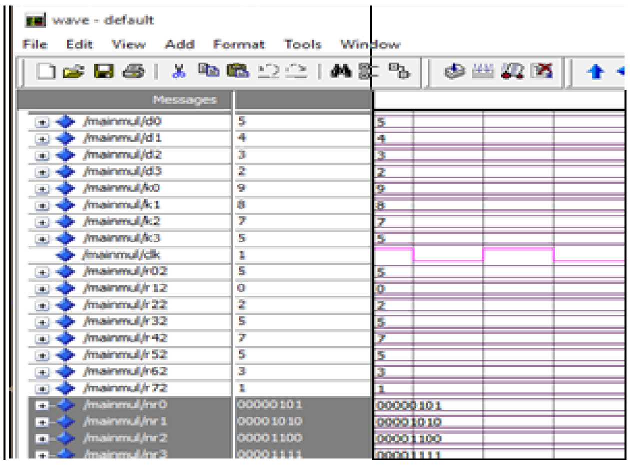
$$f_c(16) = -10^{32} + 07407407407407417037037037037037$$
$$f_c(34) = -10^{68} + 074074074074 \cdots 07417037037037.$$

5.DECIMAL PARTIAL PRODUCT REDUCTION

The PPR tree consists of three parts: (1) a regular binary CSA tree to compute an estimation of the decimal partial product sum in a binary carry-save form (S, C), (2) a sum correction block to count the carries generated between the digit columns, and (3) a decimal digit 3:2 compressor which increments the carry-save sum according to the carries count to obtain the final double-word product (A;B), A being represented with excess-6 BCD digits and B being represented with BCD digits. The PPR tree can be viewed as adjacent columns of h ODDS digits each, h being the column height (see Fig. 4), and h < = d+1.Fig. 5 shows the high-level architecture of a column of the PPR tree (the ith column) with h ODDS digits in [0, 15]. (4 bits per digit). Each digit column of the binary CSA tree (the graycolored box in Fig. 5) reduces the h input digits and n cin input carry bits, transferred from the previous

6. RESULTS& DISCUSSION

We had verified this by writing the VHDL code,we can simulated and synthesized it on FPGA board. The following results have been shown below in these two examples we have given two different values and seen the correct values. We have taken two 4 bit BCD number and performed multiplication.



```
/mainmul/d0     5         5
/mainmul/d1     4         4
/mainmul/d2     3         3
/mainmul/d3     2         2
/mainmul/k0     9         9
/mainmul/k1     8         8
/mainmul/k2     7         7
/mainmul/k3     5         5
/mainmul/clk    1
/mainmul/r02    5         5
/mainmul/r12    0         0
/mainmul/r22    2         2
/mainmul/r32    5         5
/mainmul/r42    7         7
/mainmul/r52    5         5
/mainmul/r62    3         3
/mainmul/r72    1         1
/mainmul/nr0    00000101  00000101
/mainmul/nr1    00001010  00001010
/mainmul/nr2    00001100  00001100
/mainmul/nr3    00001111  00001111
```

FIG 4. Simulation Results Of Proposed Bcd Multiplier

Table III: without mod-10 logic

Design	LUTs	Area Delay product	Power Delay product
Direct addition	1387	2064.22	0.522
Carry look-ahead adder	1405	2109.48	0.315
Manchester carry chain	2177	2948.75	0.314
Ripple carry adder	1385	2080.58	0.299

Table IV: with mod-10 logic

Design	LUTs	Area Delay product	Power Delay product
Direct addition	1570	2324.89	0.323
Carry look-ahead adder	1499	2219.81	0.317
Ripple carry adder	1495	2214.56	0.317
Manchester carry chain	2021	2744.72	0.303

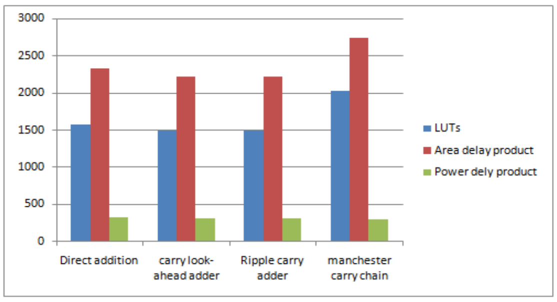


Fig 5 comparison graph without mod-10 logic

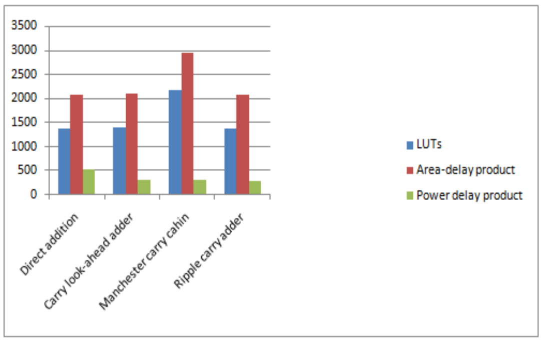


Fig6 Comparison graph with mod-10 logic

7.CONCLUSION

Finally we have observed that this product is better than older BCD multipliers. We have implemented using VHDL and simulated along with synthesis on Sparton -3e FPGA board. We have dumped into Xilinx Chip (**xcv3s400e-5s**). Thearea has been minimized by 23% which shows the decrease of power consumption by 37% than previous BCD Multiplier

REFERENCES:

[1]Alvaro Vazquez, Member, IEEE, ElisardoAntelo, and Javier D. Bruguera, Member, IEEE "Fast Radix-10 Multiplication Using Redundant BCD Codes "IEEE TRANSACTIONS ON COMPUTERS, VOL. 63, NO. 8, AUGUST 2014

[2] A. Aswal, M. G. Perumal, and G. N. S. Prasanna, "On basic finanial decimal operations on binary machines," IEEE Trans. Comput.,vol. 61, no. 8, pp. 1084–1096, Aug. 2012.

[3] 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.

[4] M. F. Cowlishaw, "Decimal floating-point: Algorism for computers," in Proc. 16th IEEE Symp. Comput. Arithmetic, Jul. 2003,pp. 104–111.

[5] S. Carlough and E. Schwarz, "Power6 decimal divide," in Proc. 18th IEEE Symp. Appl.-Specific Syst., Arch., Process., Jul. 2007, pp. 128–133.

[6] 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.

[7] L. Dadda, "Multioperand parallel decimal adder: A mixed binary and BCD approach," IEEE Trans. Comput., vol. 56, no. 10, pp. 1320–1328, Oct. 2007.

[8] L. Dadda and A. Nannarelli, "A variant of a Radix-10 combinational multiplier," in Proc. IEEE Int. Symp. Circuits Syst., May 2008, pp. 3370–3373.

[9] 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.

[10] M. A. Erle and M. J. Schulte, "Decimal multiplication via carry- save addition," in Proc. IEEE Int. Conf Appl.-Specific Syst., Arch., Process., Jun. 2003, pp. 348–358

[11] M. A. Erle, E. M. Schwarz, and M. J. Schulte, "Decimal multiplication with efficient partial product generation," in Proc. 17th IEEE

[12] Faraday Tech. Corp. (2004). 90nm UMC L90 standard performance low-K library (RVT).[Online]. Available: http://freelibrary.faraday-tech.com/

[13] 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.

[14] L. Han and S. Ko, "High speed parallel decimal multiplication with redundant internal encodings," IEEE Trans. Comput., vol. 62,no. 5, pp. 956–968, May 2013.

[15] IEEE Standard for Floating-Point Arithmetic, IEEE Std 754(TM)-2008 IEEE Comput. Soc., Aug. 2008.

[16] G. Jaberipur and A. Kaivani, "Improving the speed of parallel decimal multiplication," IEEE Trans. Comput., vol. 58, no. 11, pp. 1539–1552, Nov. 2009.

[17] R. D. Kenney,M. J. Schulte, and M. A. Erle, "High-frequency deci-mal multiplier," in Proc. IEEE Int. Conf. Comput. Des.: VLSI Comput. Process., Oct. 2004, pp. 26–29.

[18] R. D. Kenney and M. J. Schulte, "High-speed multioperand decimal adders," IEEE Trans. Comput., vol. 54, no. 8, pp. 953–963, Aug. 2005.

[19] T. Lang and A. Nannarelli, "A Radix-10 combinational multiplier," in Proc. 40th Asilomar Conf. Signals, Syst., Comput., Oct.2006, pp. 313–317.

[20] T. Ohtsuki, Y. Oshima, S. Ishikawa, H. Yabe, and M. Fukuta,"Apparatus for decimal multiplication," U.S. Patent 4 677 583, Jun.1987.

[21] R. K. Richards, Arithmetic Operations in Digital Computers. NewYork, NY, USA: Van Nostrand, 1955.

[22] M. Schmookler and A.Weinberger, "High speed decimal addition,"IEEE Trans. Comput., vol. C-20, no. 8, pp. 862–866, Aug. 1971.

[23] E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw, "Decimal floating-point support on the IBM System z10 processor," IBM J.Res.Develop., vol. 51, no. 1, pp. 4:1–4:10, Jan./Feb. 2009.

[24] B. Shirazi, D. Y. Y. Yun, and C. N. Zhang, "RBCD: Redundant binary coded decimal adder," IEE Proc. E Comput. Digit.Techn., vol. 136, pp. 156–160,Mar. 1989.

[25] H. Schmid, Decimal Computation. Hoboken, NJ, USA:Wiley, 1974.

[26] T. Yoshida, T. Maruyama, Y. Akizuki, R. Kan, N. Kiyota, K. Ikenishi, S. Itou, T. Watahiki, H. Okano, "Sparc64 X: Fujitsu's new-generation 16-core processor for unix servers", IEEE Micro., vol. 33, no. 6, pp. 16–24, Nov.-Dec. 2013.

[27] A. Svoboda, "Decimal adder with signed digit arithmetic," IEEE Trans. Comput., vol. C-18, no. 3, pp. 212–215,Mar. 1969.

[28] C. Tsen, S. Gonzalez-Navarro, M. Schulte, B. Hickmann, and K.Compton, "A combined decimal and binary floating-point multiplier," in Proc. 20th IEEE Int. Conf. Appl.-Specific Syst., Archit. Process., Jul. 2009, pp. 8–15.

[29] A. Vazquez, E. Antelo, and P. Montuschi, "A new family of high-performance parallel decimal multipliers," in Proc. 18th IEEE Symp.Comput. Arithmetic, Jun. 2007, pp. 195–204.

[30] A. Vazquez, E. Antelo, and P. Montuschi, "Improved design of high-performance parallel decimal multipliers," IEEE Trans. Comput., vol. 59, no. 5, pp. 679–693,May 2010.

[31] A. Vazquez and E. Antelo, "Multi-operand decimal addition by efficient reuse of a binary carry-save adder tree," in Proc. 44th ASI-LOMAR Conf. Signals, Syst. Comput., Nov. 2010, pp. 1685–1689.

[32] A. Vazquez and E. Antelo.(2012, Jun.).Area and Delay Evaluation Model for CMOS Circuits.Internal Report, Univ. of Santiago de Compostela, [Online]. Available: http://www.ac.usc.es/node/1607

[33] K.Suneetha, B.Rajasekharareddy, D.Raghavareddy, "FPGA Implementation of a new design in Radix-10 multiplier with Redundant BCD codes for better performance using VHDL".