JES
Journal of Engineering Sciences

Vol 13, Issue 06, JUNE/2022

ISSN NO:0377-9254

VHDL Implementation Of A New Radix-10 Multiplier Design with Redundant BCD Codes For Better Performance Using FPGA

D V GOPALA NAGASAI[1], M NAGESWARA RAO[2], S V NARESH[3]

[1]PG Scholar, Dept of ECE, SMCE , Guntur, Ap, India.

[2] Professor, Department of ECE , SMCE, Guntur, Ap, India.

[3] Professor & HEAD OF THE DEPT, Department of ECE , SMCE, Guntur, Ap, India.

ABSTRACT: The research focuses on the creation of a novel BCD parallel multiplier architecture that takes advantage of some of the qualities of two alternative redundant BCD codes to speed up computation: the redundant BCD excess- 3 code (XS-3) and the overloaded BCD representation (ODDS). We have developed several new ways to minimize the latency and area of prior sample high-performance implementations by a significant amount. Partial product generation in parallel utilizing a signed-digit radix-10 recoding of the BCD multiplier with the digit set [-5, 5], as well as a set of positive multiplicand multiples (0X, 1X, 2X, 3X, 4X, 5X) coded in XS-3, plays a major role. There are various advantages to utilizing the above method of encoding, the most important of which is that it is a self-complementing code, meaning that a negative multiplicand multiple can be generated by simply inverting the bits of the corresponding positive one. Furthermore, the available redundancy allows for the carry-free creation of multiplicand multiples, and partial products can be recoded to the ODDS representation by simply adding a constant factor to the partial product reduction tree. Because the ODDS uses a 4-bit binary encoding similar to non-redundant BCD, traditional binary VLSI circuit approaches can be used. For the final stage, we devised a novel BCD addition strategy. In comparison to older version multipliers, the above designed architecture of 4X 4 has been synthesized as an RTL model and offered higher performance.

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 useful in financial, commercial, and user-oriented computing, where conversion and rounding mistakes in floating-point binary representations are unacceptable [3]. The new IEEE 754-2008 Standard for Floating-Point Arithmetic [15], which includes a format and specification for decimal floating-point (DFP) arithmetic [1], [2], has sparked a lot of interest in decimal hardware research [6, [9], [10], [28], [30]. Furthermore, current IBM Power and z/System microprocessor families [5], [8], [23], as well as the Fujitsu Spar c X microprocessor [26], aimed at servers and mainframes, already include fully IEEE 754-2008 compliant decimal floating-point units (DFPUs) for Decimal64 (16 precision digits) and Decimal128 (34 precision digits). Since both area and power are limited 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], [19], [30] 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 [21], [25]. BCD encodes a number X in 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 more complications than binary, which lead to area and delay penalties in the resulting arithmetic units. A variety of redundant decimal formats and arithmetics have been proposed to improve the performance of BCD multiplication. 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 for carry-free accumulation of BCD partial products using rows of BCD digit adders arranged in linear [9], [20] or tree-like configurations [19]. Decimal signed-digit (SD) representations [10],[14], rely on a redundant digit set {-a ,. . . 0 . . . . a} $5 < a < 9$ to allow decimal carry-free addition.

Furthermore, these codes are self-complementing, so that the 9's complement of a digit, required for negation, is easily obtained by bit-inversion of its 4-bit representation. A disadvantage of 4221 and 5211 codes, is the use of a non- redundant radix-10 digit set [0, 9] as BCD. Thus, the redundancy is constrained to the digit bounds, so that complex decimal multiples, such as X, cannot be obtained in a carry-freeway.

In this work, we focus on the improvement of parallel decimal multiplication by exploiting 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]. We use a minimally redundant digit set for the recoding of the BCD multiplier digits, the signed-digit radix- 10 recoding [30], that is, the recoded signed digits are in the set {-5,-4,-3,-2,-1,0,1,2,3,4,5}For this digit set, the main issue is to perform the multiple without long carry- propagation (note that and are easy multiples for decimal [30] and is generated as two consecutive operations). We propose the use of a general redundant BCD arithmetic (that includes the ODDS, For this digit set, the main issue is to perform the multiple without long carry-propagation (note that and are easy multiples for decimal [30] and that is generated as two consecutive operations). 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 [30]) is that all the interesting multiples for decimal partial product generation, including the X 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 radix- 10 and BCD carry-save redundant representations require specific radix-10 digit adders [14], [22], [27].

The paper is organized as follows. Section 2 introduces formally the redundant BCD representations used in this work. Section 3 outlines the high level implementation (algorithm and architecture) of the proposed BCD parallel multiplier. In Section 4 we describe the techniques developed for the generation of decimal partial products. Decimal partial product reduction and the final conversion to a non-redundant BCD product are detailed in Sections 5 and 6 respectively.

2. Redundant BCD Representations

The proposed decimal multiplier uses internally a redundant BCD arithmetic to speed up and simplify the implementation. This arithmetic deals with radix-10 ten's

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

complement integers of the form: $Z = -s_z \times 10^d + \sum_{i=0}^{d-1} Z_i \times 10^i$, where d is the number of digits, s_z is the sign bit, On the other hand, the binary value of the 4-bit vector representation of Zi is given by $z_{i;j}$ being the jth bit of the ith 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, $Z_i = [Z_i] - e$.

Note that bit-weighted code such as BCD and ODDS use the 4-bit binary encoding (or BCD encoding) defined in Expression (2). Thus, Zi Zi for operands Z represented in BCD or ODDS. 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 straight- forward, since the digit set of BCD is a subset of the ODDS representation. In our work we use a SD radix-10 recoding of the BCD multiplier [30], which requires to compute a set of decimal multiples $(\{-5X, \ldots, 0X, \ldots, 5X\})$ 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 , r), 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 3 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

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

The implementation of 9-Zi leads to a complex implementation, since the Zi 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.

3. 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 P= 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 d-digit BCD product..

Stage 1) Decimal partial product generation. A SD radix-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 [29]. Therefore, 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 is represented in a 6–bit

hot-one code to be used as control input of the multiplexers for selecting the proper multiplicand multiple, 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 Stage 2) Decimal partial product reduction. In this stage, the array of d ODDS partial products are reduced to two d-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 ODDS partial products can be viewed as adjacent digit columns of height h d . 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).

Tree structures for the addition of ODDS operands are implemented in two previous designs [12], [18]. After all the operands in a binary CSA tree have been added, a combinational logic block is used to determine the sum correction in the non-speculative BCD adder [18], with the maximum number of inputs limited to 19 BCD operands. Our technique, on the other hand, evaluates the total correction alongside the binary carry-save additions using columns of binary counters. Basically, we count the amount of carries in each decimal column before multiplying by 6. (a correction by 6 for each carry- out from each column). The result is added to the binary carry-save reduction tree's output as a corrective term. The partial product reduction tree's latency is greatly improved as a result of this. In addition, the suggested architecture can take any number of ODDS or BCD operand inputs. Part of the PPR tree designs shown in [12] (the area-improved PPR tree) use a similar principle, although some of the stage reductions are performed by a specifically developed ODDS adder. As was done in [31] for the 4221 and 5211 decimal codings, our solution attempts to allow optimal reuse of any binary CSA tree

for multioperand decimal addition.

Stage 3) Conversion to (non-redundant) BCD. We consider the use of a BCD carry-propagate adder [29] to perform the final conversion to a non-redundant BCD product P= A+ B. The proposed architecture is a d-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 [22]. 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

4. 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 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 ; .The d least significant partial products PP d ; PP 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 invert the multiplicand multiple, to obtain its 9's complement, if the SD radix-10 digit is negative (Ysk =1).

On the other hand, if the signals are all zero then PP k , 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 Ysk Ysk Ybk is zero (denotes the boolean OR operator), where Ybk iszero 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 (see Section 4.2). The generation of the most significant partial product PP and only depends on Ysd , the sign of the most significant SD radix-10 digit.

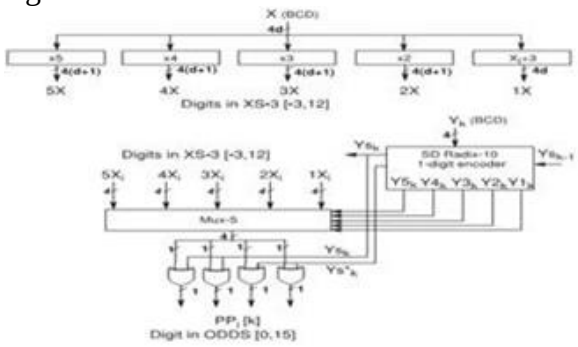

Fig1:SD Radix 10 generation of partial product digit Generation of the Multiplicand Multiples

We denote by NX 1X; 2X; 3X; 4X; 5X , the set of multiplicand multiples coded in the XS-3 representation, with digits NXi ; , being NXi NXi ; the corresponding value of the 4- bit binary encoding of NXi given by Equation (2).Fig. 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 adecimal carry and a digit such as being Tmax 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 NXi, that is

$$(Y1_k, Y2_k, Y3_k, Y4_k, Y5_k)$$

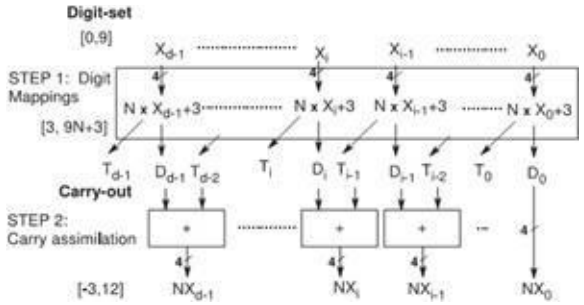

Fig 2: Generation of decimal multiplies

$[NX_i] = D_i + T_{i-1}, [NX_i] \in [0, 15](NX_i \in [-3, 12]).$
Most-Significant Digit Encoding The MSD of each PP k , PPd 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

	1.X			2.X			3.X			4.X			5.X		
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

Table 1: Prefer Reading Mapping Values
The resultant partial product sum has to be corrected off the-critical-path by adding a pre-computed term, f_c which only depends on the format precision d. This term has to gather: (a) the constants that have not been included in the MSD encoding and (b) a constant for every XS-3 partial product digit (introduced to simplify the nine's complement operation). Actually, the addition of these 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.

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

$$f_c(16) = -10^{32} + 0740740740740741703703703703703 7$$
$$f_c(34) = -10^{68} + 074074074074 \cdots 07417037037037.$$

5. Decimal Partial Product Reduction

The PPR tree is divided into three sections: (1) a regular binary CSA tree to estimate the decimal partial product sum in 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 to increment the carry-save sum according to the carries count to obtain the final double-word product (A;B), where A is represented with excess-6 BCD digits and B is represented with BCD digits. The PPR tree can be visualized as a series of contiguous columns, each with h ODDS digits (see Fig. 4), with h = d+1. The high-level architecture of a PPR tree column (the ith column) with h ODDS digits in [0, 15] is shown in Fig. 5. Each digit has four bits. The h input digits and n cin input carry bits, transferred from the previous digit column of the binary CSA tree (the grey coloured box in Fig. 5) are reduced in each digit column of the binary CSA tree (the grey colored box in Fig. 5).

6. RESULTS

We confirmed this by developing VHDL code and simulating and synthesizing it on an FPGA device. We've given two distinct values in these two examples and observed the proper values. Multiplication was performed on two four-bit BCD numbers.

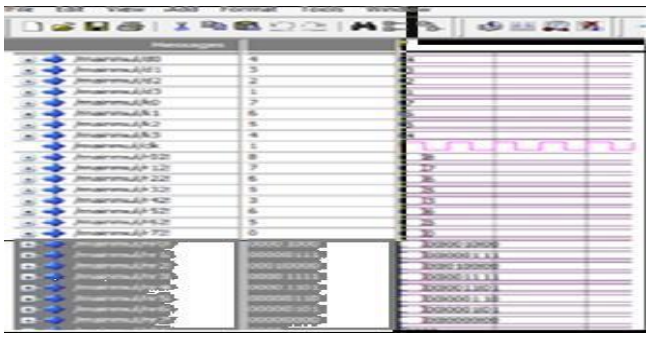

FIG 3: Simulation Results Of Proposed BCD Multiplier

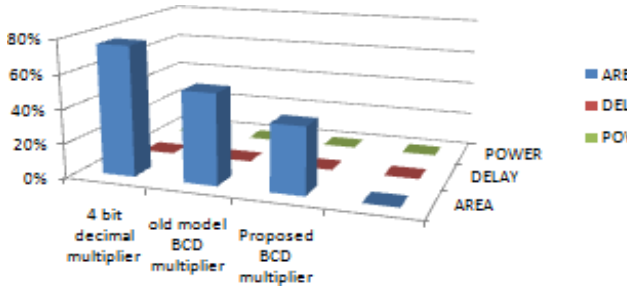

FIG 4 Comparison graph of Area, Delay and Power

7. CONCLUSION

Finally, we discovered that our product outperforms previous BCD multipliers. On the Sparton -3e FPGA board, we implemented using VHDL and simulated along with synthesis. We've dumped everything into a Xilinx Chip (xcv3s400e-5s). The area has been reduced by 23%, resulting in a 37% reduction in power consumption over the previous BCD Multiplier.

REFERENCES:

1. Alvaro Vazquez, Member, IEEE, Elisardo Antelo, 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.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

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

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.
13. 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.