International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 9, September - 2013

Design Implementation of an Efficient BCD Multiplier in a Fully Pipelined Structure

J.Shankar Rao [1], A.Suman Kumar Reddy [2]
[1]*Student of M.Tech. (VLSI), PBR VITS, Kavali, AP, India*
[2]*Associate Professor, Department of ECE, PBR VITS, Kavali, AP, India*

Abstract—*Decimal multiplication is one of the most frequently used operations in financial, scientific, commercial and internet based applications. This paper presents an efficient implementation of a fully pipelined decimal multiplier designed with Carry Save Addition and coded into a reduced group of BCD-4221. This design is based on multiplier operands recoded in Signed-Digit radix-10, a simplified partial products generator, and decimal adders.. Several assessments are carried out in various N by M multiplications and their respective synthesis results show slightly optimistic figures in terms of area and delay in regard to some previously published works.*

Keywords*-FPGA; BCD; Computer Arithmetic; decimal floatingpoint; signed-digit radix-10.*

1. INTRODUCTION

Decimal multiplication plays a key role in many applications. This is the reason why over recent years decimal operations have become very popular. The major advantage is the greater precision respect to binary systems. Currently, this entails the development of high-performance binary-decimal arithmetic circuits

There are several works which focus on fixed-point multiplication. Earle and Schulte propose two novel sequential designs for fixed-point decimal multiplication developing a decimal carry-save addition in order to reduce the critical path delay. Decimal sequential multiplier has been developed by Sutter et al. [6] and described important features of efficient decimal multipliers using either embedded BRAM´s or a low level implementation with LUTs, MUXFX multiplexers, and the usage of fast adders. An implementation of a decimal parallel multiplier .

This work presents the algorithm, architecture and implementations of an efficient pipelined multiplier. Its development is based on carry save addition (CSA) techniques in order to compress the partial products tree.

2. AN OVERVIEW OF BCD MULTIPLIER

A generic N-digit by M-digit multiplication P = AxB is indicated in Fig. 1, where A and B are the multiplicand and multiplier respectively. The operation is made of the following modules: generation of partial products, reduction of partial products tree, and fast decimal adders. Decimal multiplication coded in BCD-8421 (8421) presents a more complex implementation than binary multiplication due to the presence of invalid (8421) digits between {A, B, … F}. These need to be corrected generating an extra cost in computation as well as additional multiplicand multiples that must be implemented by the multiplier [8].The generation of decimal partial products is based on the techniques described in [8, 3]. First of all, decimal digits of A are represented in BCD-4221 (4221) to prevent the corrections previously mentioned. This reduces the computation complexity of multiplicand multiples coded into (4221). A BCD-5211 (5211) format is introduced in this Section. Both of these formats are necessary to determine the set of multiples. For example, the multiple 2A is computed as follows: each(8421) digit is first recoded to (5421) decimal as it is shown in Fig. 2a; straightaway a 1-bit left shift is carried out, obtaining the 2A multiple in (8421). The 2A multiple is easily recoded from (8421) to (4221). The rest of multiples are computed according to Fig. 2a. The multiplier B is recoded into signed-digit

(SD) radix-10. The (4221) or (5211) coding is convenient because speed-up the CSA eduction, whose 9´s complement is generated by simple bit inversion.

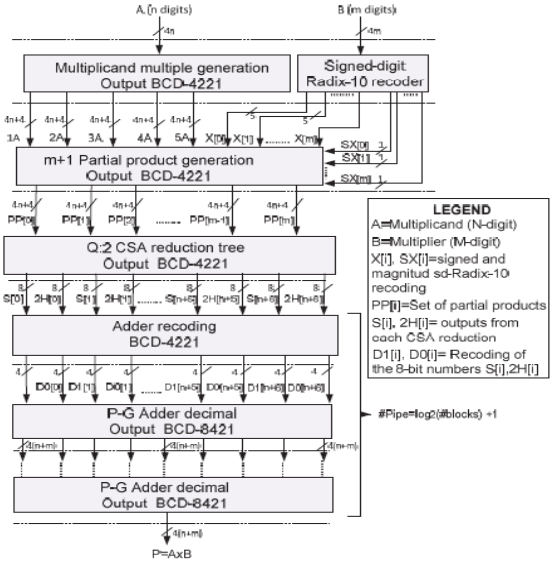
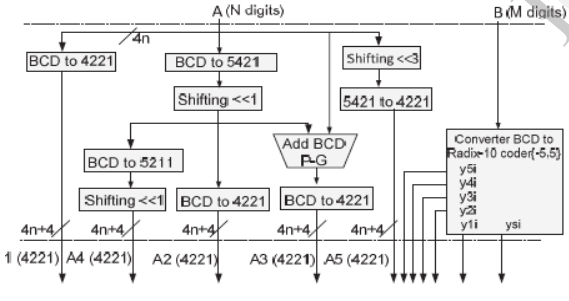


Figure 1. Decimal Sd Radix-10 pipelined multiplier diagram this paper proposes decimal Q:2 CSAs which reduce Q digits to two ones. These reductions are implemented in different versions, for Q = 3, 4, 5, 8, 9 decimal operands.



a) Multiplicand Multiples Generation

N	(4221)	Reduced (4221)
0	0000	0000
1	0001	0001
2	0100, 0010	0100
3	0101,0011	0101
4	0110,1000	0110
5	1001,0111	1001
6	1010,1100	1010
7	1011,1101	1011
8	1110	1110
9	1111	1111

b) Reduced BCD-4221 coding

Figure 2. Multiples for SD radix-10 encoding This will be further detailed in Section 3. It must be pointed out that the processing is based on the (4221) coding. Several decimal additions are applied over two reduced digits. The fast carry-chain adder described in [10] is a suitable alternative due to its high performance in terms of area-delay.

3. NXM BCD MULTIPLIER IMPLMENTATION

A general overview of NxM-digit BCD multiplier architecture is described below and depicted in Fig.1. Dashed blocks indicate the main odules of the design, and the dotted line indicates the pipelined stages.A N-digit multiplicand A and a M-digit multiplier B as unsigned decimal are assumed. The (8421) multiplication is processed as follows: the multiplicand multiples generation unit takes the operand A and computes a set of N + 4-digit multiplicand multiples {1A, 2A, 3A, 4A, 5A} coded into (4221). In parallel, the sd-radix-10 unit recodes each digit (8421) of B between {-5, -4, -3, … +3, +4, +5} that is represented with a 1-bit sign and 5-bit magnitude format. The recoded B is multiplied digit-by-digit by the previously computed multiplicand multiples. It is important to emphasize that the (8421) coding introduces a computational cost due to the corrections in the decimal reduction CSA. An alternative toprevent this drawback is to use (4221) coding [8].Immediately after, the partial product generation unit takes the outputs from the two modules previously mentioned and generates M + 1 partial products which depend on eachrecoded digit of B, represented in signed-digit radix-10 format.Each partial product coded in (4221) is at most of N + 3-digit length. All of partial products are taken as inputs in the decimal Q:2 CSA reduction tree module, this simplifies Q-digit into two decimal digits coded in (4221). The basic scheme with regard to decimal 3:2 CSA reduction fulfils the relation A(i) + B(i) + C(i) = 2H(i) + S(i), i ∈ □{0, 1, 2, … M - 1} (Fig. 3a). The 4-bit inputs A(i), B(i), and C(i) are simplified to 4-bit S and H by means of 4 full-adder cells. It can be noted that the 2H(i) is produced by the block 2X. therefore, the considered outputs will require at least 5-bit operands. The proposed decimal Q:2 reduction presents two outputs: 2H and S of 8 bits each one. This architecture can be extended to different versions of decimal Q:2 compressors. To validate and carry out the evaluations, several decimal Q:2 CSAs between 3:2 and 9:2 compressors were tested. The proposed decimal Q:2 CSA is base on several basic binary CSAs (Fig. 3b, 3c, 3d, 3e) and these equally are made up of full or half-adders as basic cells. . The early block is based on a(4221)

addition and a circuit to detect special cases. Straightaway, the (4221) outputs are recoded into (8421).Each mentioned stage pipeline executes a decimal addition which is based on carry-chain techniques. Finally, the result from the last pipelined stage shows the multiplication N + Mdigit AxB

.

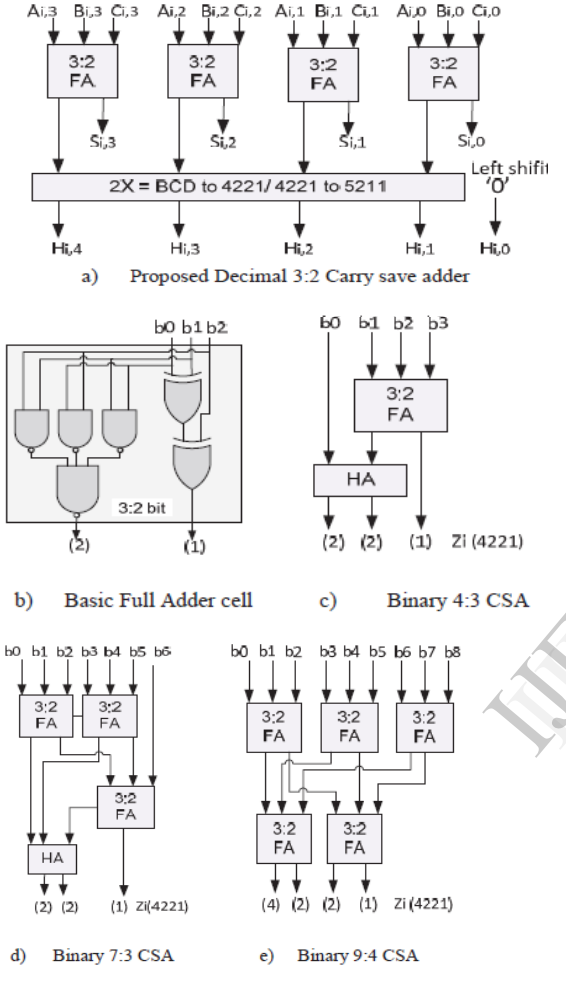


Figure 3. Proposed Decimal 3:2 CSA and several Binary Adder schemes

A. Signed-Digit Radix-10 recoding / Multiplicand Multiples Generation

The block diagram is depicted in the Fig. 2a, the recoding is applied to each digit of B. This transforms a (8421) set {0, 1, 2 .., 9} into the signed-digit set {-5, -4… +4, +5} made up of a sign signal SX(i) and a 5-bit magnitude X(i). Each digit X(i) selects the corresponding multiplicand multiples coded in (4221). The sign SX verifies if a negative multiple is produced. It is important to highlight that a negative version of a partial product is obtained simply by inverting the bits of the positive version using arrays of XOR gates manipulated by SX(i). The above circuit requires 2 slices and 6 LUTs.The following section discusses Multiplicand Multiples Generation (Fig. 2a), this set of multiples are coded into (4221). Multiples 2A and 5A are generated with recoding (8421 to 4221, 5211 to 4221) and carrying out a left shifting process. Multiple 4A is computed as 2x2A and 3A comes from a decimal addition between A and 2A. Each partial product has N + 3 digits.

B. Reduction partial product

After generating the M + 1 partial products, coded into (4221), the reduction of partial products is developed by means of decimal Q:2 carry save additions (CSA). First, the partial products are aligned and divided into blocks as it is shown in Fig 4. A circuit based on decimal 3:2, 4:2, and 8:2 CSA compressors is proposed, which entail to utilize block sizes of (M + 3 + M / #blocks) x Q decimal operands

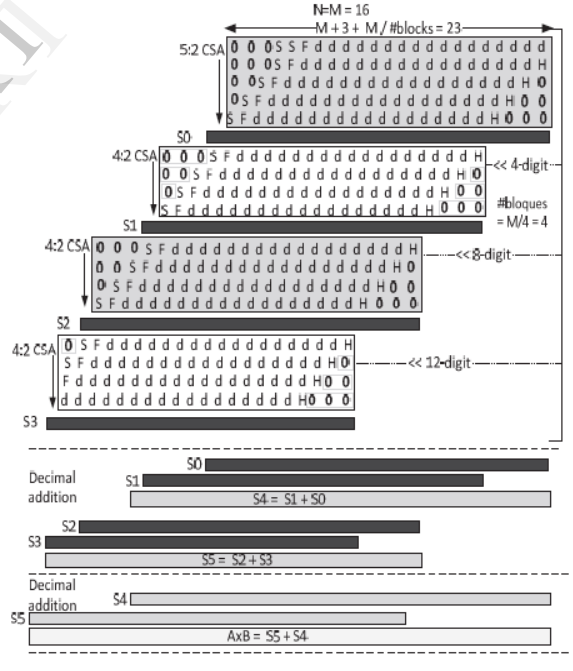


Figure 4. Example of a partial produts reduction architecture

A decimal 3:2 CSA compressor (Fig. 3a) adds three numbers A, B, C each one of size 4-bit (coded in 4221) to get a decimal S and a carry H, such that A + B + C = S + 2H as it was explained in Section 3. The 2X module observed in Fig. 3a is composed of a (4221 to 5211) coding operation and a one bit left shift operation. The decimal 3:2 CSA is implemented by 4 binaries 3:2 CSA, each one

utilizing 2 LUT3s, and a 2X block. The hardware cost of different decimal Q:2 versions are shown in the Table I. The above basic binary 3:2 (full adder) circuit is utilized as basic cell in different CSA compressor schemes. As an example the decimal 8:2 CSA will be described: This
implementation requires 4 binary modules 8:4 CSA, 2 binary 3:2 CSA and seven 2x blocks. Each binary 8:4 CSA is made up of one half and 4 full adders and has an area of 7 LUT4s and 3 LUT6s. Table I exhibits the area contribution of several decimal CSA implementations

TABLE I. AREA OF SEVERAL PROPOSED DECIMAL CSAS

Decimal Q:2	*LUT3s*	*LUT4s*	*LUT5s*	*LUT6s*
4:2	16	12	-	-
5:2	16	-	12	
6:2	16	16	-	12
8:2	32	28	-	12
9:2	32	28	16	12

C. BCD-4221 Adder recoding

After processing the earlier stage, the (4221) Adder Recoding unit in Fig. 5 takes as inputs the outputs of each Q:2 reduction and codes them into two decimal digits (4221) of 4-bit size. This implementation is based on a (4221) addition and a circuit to detect special cases. As it is specified in the Section
3, a (4221) decimal addition does not require correction as usually occurs in the (8421) decimal adder. As soon as the circuit receives the two outputs of 8 bits of the early implementation, a 2-digit decimal addition in (4221) format is carried out. The function is represented as: 2H(i)(7..0) + S(i)(7..0) = D1(i)(3..0) & D0(i)(3..0), where (i) represents the “i-th” Q:2 reduction process. In Fig. 5 the implemented architecture can be observed.

D. Decimal Addition design

After finishing the previous stage and as soon as its outputs are computed, a (8421) decimal addition is carried out. First of all, an aligned process on the vectors D0 and D1 is eveloped,just before both of them are recoded into (8421). As it is pointed out at the beginning of this section, various levels of pipeline = og2 (M) + 1 are implemented.

Several decimal additions are processed at each pipelined stage (Fig. 4),carrying out additions of (M + 3 + M / #blocks) digits.The proposed addition is based on 10´s complement BCD numbers, and the carry-chains techniques to carry out the full design.

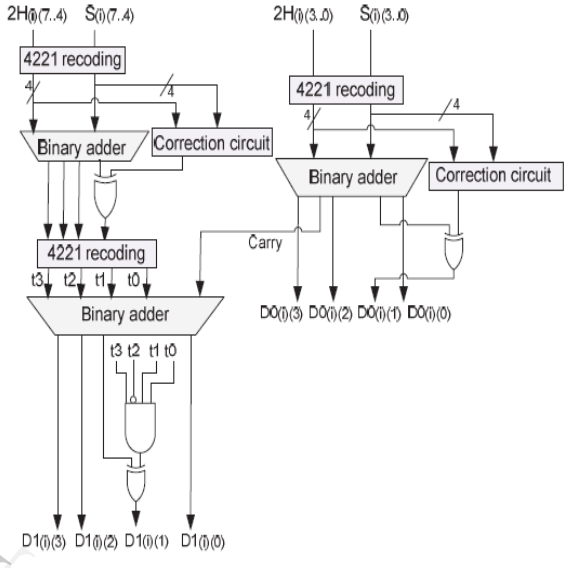


Figure 5. BCD-4221 Adder recoding architecture

TABLE II. IMPLEMENTATION RESULT OF NXM DECIMAL MULTIPLIERS FULLY PIPELINED

N	*M*	*#LUTs*	*#Slices*	*#FFs*	*F (Mhz)*	*#Pipe*	*Delay (ns)*
4	4	634	205	368	421.9	5	11.8
8	4	1084	396	609	408.1	5	12.25
16	4	2021	703	1106	370.4	5	13.5
32	4	3873	1281	2090	344.8	5	14.5
4	8	1191	380	696	392.2	6	15.3
8	8	2109	668	1111	374.9	6	16
16	8	3815	1214	1939	344.8	6	17.4
32	8	7289	2539	3583	313.5	6	19.1
4	16	2379	791	1370	373.1	7	18.7
8	16	4107	1300	2126	344.8	7	20.3
16	16	7312	2262	3618	322.6	7	21.7
32	16	14352	4571	6643	287.4	7	24.3
4	32	4882	1641	2800	370.4	8	21.6
8	32	8044	2635	4206	322.6	8	24.8
16	32	14616	4495	7052	270.3	8	29.6
32	32	28170	8250	12758	243.9	8	32.8

The circuits previously mentioned were built using different techniques as was explained in Section I, but it can be noted that the proposed circuit presents a delay of almost 18% and 28% less than the circuits in [4] and [6] respectively. Due to partitioned scheme implemented a large area penalty is generated

by the proposed multiplier in regard to equentialone in [6].

IV.(a). Results verification

To validate the designs, large numbers of random vectors were applied to an automated testbench that tests the behavioural models using ModelSim. For the decimal multiplier, over 10,000 test cases were used. A pipelined multiplier with generic width and depth was implemented, which carries out the above mentioned tests. Additionally a random numbers generation module was implemented and used to validate the results.

IV(b).Results Comparison

As a first comparison, the proposed design matches up with three different decimal multiplication implementations on FPGAs reported in [6, 9, 11]. Table III shows the figures in terms of area and delay for 16x16 decimal multiplications, as well as a binary multiplier provided by Xilinx Core Generation.
It presents the performance of several multiplications implementations on Virtex-4 [6, 11] and Virtex-6 with speed grade-3 [9] highlighting that our proposal was implemented with speed grade-2. It is worth pointing out that any comparison between circuits implemented on different FPGAs is strongly unfair.It can be deduced that the proposed scheme presents delays comparable with the binary multiplier and the proposal implemented in [9]. The advantage is that our design requires 47% less FFs but increases the number of LUTs utilized by almost 20% compared with the proposal in [9]. It is worth mentioning that the reduction tree proposed by Vazquez et al.is built on Carry-Ripple adder techniques and the multiplier operand is coded using the Signed-digit radix-5 recoding which is fully mapped into LUT-4s and LUT-6s [9]. Furthermore,
they show that the hardware cost of the previous scheme is more efficient than the Signed-Digit Radix-10 recoding utilizedby our proposal.

TABLE III. DELAY AND AREA OF 16X16 DECIMAL MULTIPLIERS IMPLEMENTED ON DIFFERENT PLATFORMS

Multiplier	*F (Mhz)*	*# FFs*	*# LUTs*	*Other Resources*	*# Cy*	*Delay (ns)*
Vazquez et al. [9]	453	6926	5008	-	8	17.7
Xil. Core 53x53 bits	450	276	133	10 DSP	12	18.5
Sutter et al. [6]	154	459	1045	8 BRAM	17	110.5
Vestias w/DSP [11]	-	-	3035	16 DSP	-	68
Vestias wo/DSP [11]	-	-	6493	-	-	65
Proposed multiplier	344.8	3643	6273	-	8	23.2

A Second comparison is presented in table IV which shows delay and area for each NxM multiplication achievement for different
the 8x8- and 16x16-digit multipliers present an optimal pipeline depth of 6 and 7 respectively presenting the lowest delays but with penalization in area.

TABLE IV. DELAY AND AREA OF 16X16 DECIMAL MULTIPLIERS IMPLEMENTED ON DIFFERENT PIPELINED LEVELS

N	*M*	*#PiPe*	*#LUTs*	*#Slices*	*#FFs*	*F (Mhz)*	*Delay (ns)*
8	8	5	1897	629	899	289.9	17.2
8	8	6	2109	668	1111	374.9	16
8	8	7	1629	572	1003	384.6	18.2
16	16	6	6365	2182	2840	263.9	22.7
16	16	7	7312	2262	3618	322.6	21.7
16	16	8	6273	1941	3643	344.8	23.2

TABLE V. DELAY AND AREA OF 16X16 DECIMAL MULTIPLIERS IMPLEMENTED ON DIFFERENT FPGAS

FPGA	*Mult. NxM*	*#LUTs*	*#Slices*	*Delay (ns)*	*F (Mhz)*
xc6vlx760 ff1760 -2	4x4	634	205	11.9	421.9
	8x8	2109	668	16	374.9
	16x16	7312	2262	21.7	322.6
	32x32	28170	8250	32.8	243.9
xc5vlx330 ff1760 -2	4x4	628	219	14.4	347.2
	8x8	2084	705	20.4	294.1
	16x16	7509	2542	26.6	263.2
	32x32	28487	9294	36	222.2

Finally, Table V provides performance comparisons between multiplications implemented on two large FPGAs,Virtex-5 and Virtex-6. Taking the 8x8-digit multiplicationpattern into consideration can be noted that in a single Virtex-5 (lx330) can fit 73 operation cores meanwhile in a Virtex-6 (lx760) fits 177 ones, the figures demonstrate the highperformance of our proposal.

V. CONCLUSION

This paper has reassessed several implementations of N- xM-digit multiplications on Xilinx FPGAs. This work presents the design of several BCD multipliers and their implementations on Virtex-6 FPGA. Previous techniques and designs proposed are analyzed to carry out erformancecomparison in terms of area-delay as it was seen in Section 4.The proposed pipelined multiplication is based on a multiplier operand coded into SD radix-10 recoding. A parallelgeneration of decimal partial products is reduced by carry save adder techniques and decimal adders.The proposed circuit presents a high-performance design of decimal multiplier. Our proposal shows encouraging results:one hand, its figures are comparable and outperformed than other multipliers. The other hand, a considerable number of cores can be fit into large FPGA. Future work plans to include the proposed multiplier in operations based on decimal floating point.Also, a good alternative would be to code the multiplieroperand into SD radix-5 and SD radix-4 formats and to compare the performance results. Finally, a Xilinx Microblaze Embedded processor can be utilized to execute multiplications on SW and HW, in order to estimate the SW-HW performancecost.

REFERENCES

[1] IEEE Standard for Floating-Point Arithmetic, 2008. IEEE Std 754-2008.

[2] M. A. Erle and M. J. Schulte. Decimal multiplication via carry-saveaddition. In *Proc. IEEE Int Application-Specific Systems, Architectures,and Processors Conf*, pages 348–358, 2003.

[3] B. Hickmann, A. Krioukov, M. Schulte, and M. Erle. A parallel IEEEP754 decimal floating-point multiplier, 2007. Computer Design, 2007.ICCD 2007. 25th International Conference on.

[4] H. C. Neto and M. P. Vestias. Decimal multiplier on FPGA usingembedded binary multipliers. In *Proc. Int. Conf. Field ProgrammableLogic and Applications FPL 2008*, pages 197–202, 2008.

[5] E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw. Decimalfloating-point support on the IBM System z10 processor, 2009. IBMJournal of Research and Development.

[6] G. Sutter, E. Todorovich, G. Bioul, M. Vazquez, and J.-P. DeschampsFPGA Implementations of BCD Multipliers. In *Proc. Int. Conf.Reconfigurable Computing and FPGAs ReConFig '09*, pages 36–41,2009.