Indian Journal of Science and Technology, *Vol 8(19), DOI: 10.17485/ijst/2015/v8i19/77160, August 2015* *ISSN (Print) : 0974-6846* *ISSN (Online) : 0974-5645*

Implementation of Fast Radix-10 BCD Multiplier in FPGA

H. A. Anil Kumar* and S. Umadevi

School of Electronics Engineering (VLSI Design), VIT Chennai Campus, Chennai - 600127, Tamil Nadu, India; anilkumar.ha2013@vit.ac.in

Abstract

Background: Multiplication is the basic operation in any signal processing systems and financial applications, all these applications requires multiplication to be performed in a faster and efficient manner on a silicon chip. **Methods:** This paper describes the algorithm and architecture of a BCD parallel multiplier. The design exploits two properties of redundant BCD codes to speed up its computation. Namely, the redundant BCD excess-3 code (XS-3), and the overloaded BCD representation (ODDS). In addition to this, number of new techniques are used in order to reduce significantly latency, area and for implementation on FPGA compared to existing implementations. **Findings:** Parallel architecture is used for generating partial products using radix-10 recoding technique for signed-digit of a BCD multiplier having set of digits between the range [–5, 5] and a positive set of multiplicand multiples coded in XS-3. Use of this encoding has various advantages like, as XS-3 is a self-complementing code, finding a negative of it is by just complementing the bits of respective number. Also the redundancy in XS-3 code is utilized for generating multiplicand multiples in a simple, faster and a carry-free way. Implemented design has three stages. Partial product generation, reduction and final conversion to BCD. For to implement the design in hardware the partial product reduction architecture is modified here to use a bank of ripple carry adder trees. ODDS representation uses 4-bit binary encoding technique which is similar to non-redundant BCD code, for this reason conventional VLSI circuit techniques such as carry-save adders and compression trees can be used effectively to perform operations on decimal numbers. **Conclusion:** To show the advantages of the resulted design, RTL model for 8×8-digit and 16×16-digits multiplication has been synthesized and implemented in Virtex-5 FPGA device. Results shows that the multiplier is about 10–15% delay efficient with existing work and about 14–18% area efficient.

Keywords: Excess-3, FPGA, ODDS, Overloaded BCD, Radix-10 Multiplication

1. Introduction

Multiplication of decimal numbers plays a vital role in many user-oriented applications like finance and commercial and where rounding and conversion of numbers those inherent to binary representations in floating-point cannot be tolerated[4]. This is the cause for decimal operations to become more popular in the recent years. The existence of various microprocessors like Fujitsu Sparc X and IBM power microprocessor[6,9] families that are oriented to mainframes and servers include fully IEEE 754-2008 Decimal Floating Point Units (DFPUs) for 16-digit decimal and 32-digit decimal formats. Also, the standard IEEE 754-2008 for Floating-Point Arithmetic which includes specifications and format for decimal multiplication have created a path for decimal hardware[7,10,11] in a significant manner. Again, division and multiplications are performed by using digit by digit algorithm in iterative manner because of which it adds to low performance. The use of aggressive cycle time in Processors will employ additional constraints on parallel techniques in order to reduce the latency employed by parallel design[7]. Thus, an efficient algorithm is required to develop a DFPU to perform multiplication and to have a most regular VLSI layouts which allows pipelining effectively.

The hardware implementation for BCD is more widely used as fixed point values than the binary numbers for easy conversion between user representation and the machine. However the use of decimal numbers has

**Author for correspondence*

its own disadvantages. Decimal number is represented using four bit binary number ranging from [0–9], since the digits from 10 to 15 are left out in BCD it raises the complications than the binary number system and adds to the delay and penalties in terms of area required for arithmetic units.

A variety of designs had been proposed for designing a BCD multiplier to improve its performance of multiplication operation. The carry save format[10] used in BCD multiplication represents a radix-10 operand using a carry value at its each decimal position. The use of carry-free accumulation of partial products resulted from BCD operands requires a series of BCD digit adders[10] or a tree configuration of adders[1]. The use of decimal signed-digit representation[11,15] relays on digit set of redundant values {-x,….,0,….,x}, $5 \leq x \leq 9$, to allow carry free addition. The available radix-10 with signed-digit and carry save arithmetics offers improvements in performance. However this type of VLSI implementation techniques will result in irregular layouts than the existing binary carry save adders and compressor trees.

Various redundant BCD codes namely, 4221, 5211 and 4211 can be used in BCD arithmetic as their 4-bit decimal codes satisfy the sum of weights of bits to be equal to 9, so that all the combinations of a 4-bit number can be used for a decimal [0–9] number representation. This property of redundancy can be used to speed up decimal operations without altering the data path width. Also the main advantage of using redundant numbers is in getting complement of a number, which is obtained by just inverting the bits of a 4-bit representation. A disadvantage of such codes is that it is constrained to digit bounds, due to which obtaining the multiple 3X is not possible in a carry-free way.

The ODDS (overloaded BCD or overloaded decimal digit set) representation was proposed to improve the decimal operation in parallel[13,14] and sequential decimal multiplication. These codes represent the redundant radix-10 digit $X_i \in [0, 15]$. The use of ODDS code has various advantages in hardware implementation namely: (1) it allows us to generate simple and complex multiples (3X, 4X, 5X…) and to perform addition in a carry-free way, (2) Unlike in simple BCD there is no need of additional hardware to correct the invalid 4-bit combination and (3) Since digits are represented in binary format, there is no need for extra hardware other than binary arithmetic circuits.

In this paper the algorithm, architecture and FPGA realization of BCD multiplier which focuses on improving the multiplication operation using parallel architecture by utilizing the redundant property of two decimal representations: that is, redundant BCD excess-3 (XS-3) code and the ODDS. The minimal number of digits is used for recoding of the decimal numbers. The signed-digit radix-10 digits used here are in the range of {–5,–4,…., 0,…,4,5}. Main issue with this set of digits is obtaining of multiples without long carry propagation. In this proposal acceleration to the multiplication operation is given in two steps: Partial Product Generation (PPG) and Partial Product Reduction (PPR) which are explained in coming sections.

2. Redundant BCD Representations

The proposed BCD multiplier uses a redundant number internally to simplify the implementation and to increase the speed of operation. The radix-10 ten's complement has been used which is described below:

$$Z = -S_z \times 10^n \sum_{x=0}^{n-1} Z_x \times 10^x \tag{1}$$

Where n being number of digits, $Z_i \in [m - e, 1 - e]$ is the ith digit with range from $0 \leq m \leq e$, and $9 + e \leq m \leq 2^4 - 1$ (=15) and S_z is the sign bit. Parameter e is the excess value which takes the range from 0 (for non-excess) 3, 6.

In this work, signed digit radix-10 is used for a BCD multiplier to compute the multiplicand multiples. The main issue with the multiplication (mainly for 3X) is to avoid the carry propagation for a longer path. The range of digits obtained after generating multiple 3X is in between [0, 11] hence maximum carry that is obtained will be of just one digit. The nine's complement of a positive decimal number is given by the equation,

$$-10^n + \sum_{i=0}^{n-1} (9 - Z_i) \times 10^i \tag{2}$$

The implementation of the term $(9 - Z_i)$ has more complexity as the number exceeds 9. So the implementation can be made simpler by taking excess-3 value of nine's complement which is obtained by just taking bit complement of excess-3. Table 1 shows how to get the nine's complement.

3. Design Methodology

The design methodology for a p × q-digit BCD multiplier is shown in Figure 1. This design accepts inputs that is conventional decimal such as redundant BCD operands M, N and produces a BCD product after generating a partial products array (redundant). The following three stages of operation has to be performed to obtain the decimal product: (1) Partial product generation (PPG) along with generation of multiplicand multiples coded in excess-3 and recoding of multiplier operand. (2) Partial Product Reduction (PPR) after recoding of partial

Products to ODDS code and (3) Conversion to non-redundant BCD product of p + q digits. The detailed explanation of design methodology is given in section 3.1, 3.2 and 3.3.

3.1 Decimal Partial Product Generation

A signed-digit radix-10 recoding is used in the proposed BCD multiplier. This type of design reduces the number of partial products generated so that the area of the physical layout required is reduced. Generation of partial products include recoding of multiplier digit to signed-digit radix-10, calculation of multiples in excess-3 code and generation of ODDS partial products. Figure 2 shows architecture for generating partial products. The multiplicand digit Y_k of a BCD number is recoded in SD radix-10 recoder to generate a 5-bit one-hot code represented by $Y_{bk} = \{Y1_k, Y2_k, Y3_k, Y4_k, Y5_k\}$. The obtained Y_{bk} is used as a select lines for a 5:1 multiplexer to select a respective multiplicand multiples {1X, 2X, 3X, 4X, 5X} Y_{bk} also include 1-bit value called signed bit, which is used to control the negation of a selected number.

3.1.1 *Multiplicand Multiples*

Set of multiplicand multiples are generated by coding in excess-3 format given by NX ϵ { 1X, 2X, 3X, 4X, 5X } with digits NX_i in the range [−3, 12], defined by $[NX_i] = NX_i + 3$, ϵ [0, 15]. As shown in Figure 3, it takes two steps:

Table 1. Nine's complement for XS-3 representation

Digit 4-bit encoding Z_i [W_i]	Nine's complement 4-bit encoding $9 - Z_i$ $15 - W_i$
0000 −3 0	1111 12 15
0001 −2 1	1110 11 14
0010 −1 2	1101 10 13
. . .	. . .
. . .	. . .
. . .	. . .
1101 10 13	0010 – 1 2
1110 11 14	0001 – 2 1
1111 12 15	0000 – 3 0

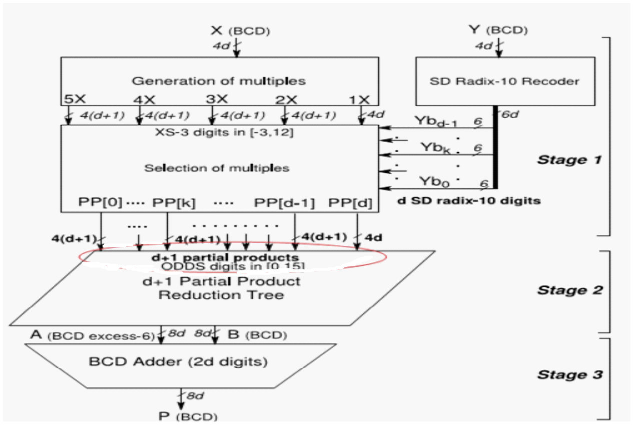


Figure 1. Signed-digit BCD Radix-10 multiplier.

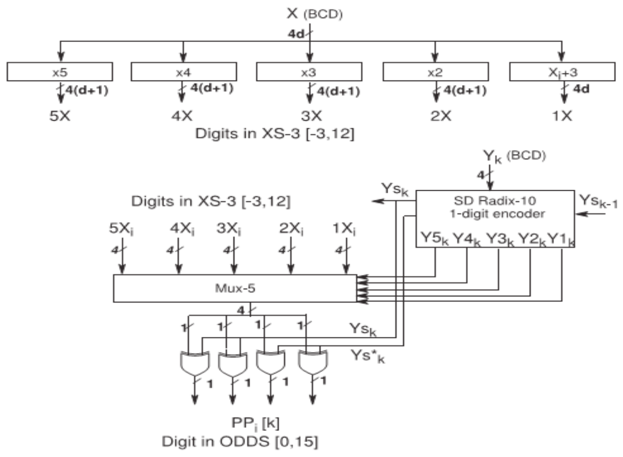


Figure 2. Generation of SD radix-10 partial product.

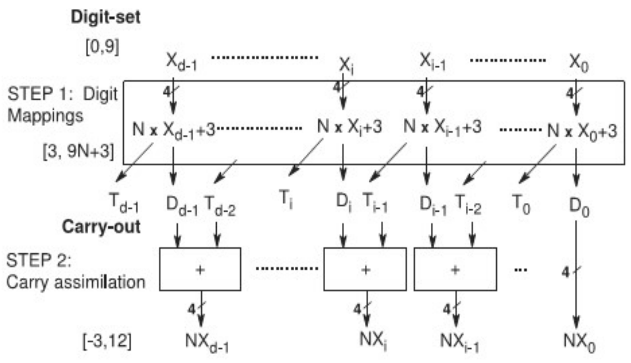


Figure 3. Decimal multiples NX generation.

- Digit Mapping of multiplicand digits and
- Assimilation of the carries between adjacent digits.

3.2 Decimal Partial Product Reduction

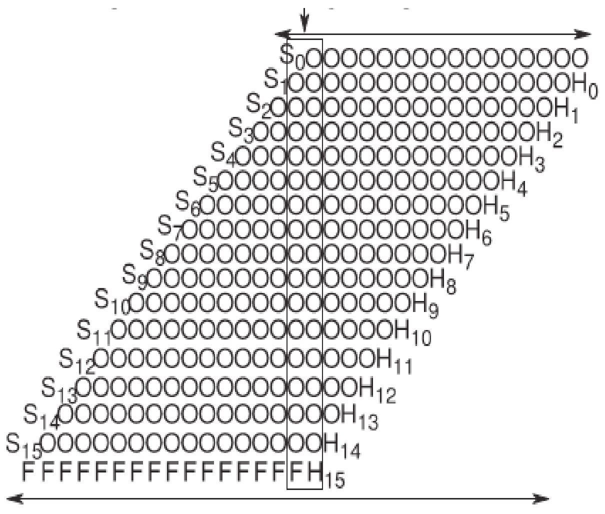
3.2.1 *Existing PPR Tree*

Decimal PPR tree is used here to reduce the number of partial products generated. The obtained partial products in the PPG step are maintained in the form an array as shown in Figure 4. Where S_k is the sign-bit encoding, O is the ODDS digit in the range [0, 15], H_k is the ten's complement encoding given by: Y_{Sk} + {0, 3, 7} and F is the ODDS digit in range [0, 15]. The high level architecture of the PPR consists of three parts: (1) A regular binary Carry save adder tree to compute an estimation of decimal partial product sum in a binary carry save form (S, C), (2) Sum correction block to count the carries generated between 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) where 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 and h ≤ d+1.

Figure 5 shows the high level architecture for existing PPR unit.

3.2.2 *Drawbacks of Existing Design*

The presence carry-save adder tree in the existing architecture reduces the efficiency of a multiplier. Carry-save adder trees requires additional logic for correction



S_0OOOOOOOOOOOOOOOOOO
S_1OOOOOOOOOOOOOOOOOOH_0
S_2OOOOOOOOOOOOOOOOOOH_1
S_{15}OOOOOOOOOOOOOOOOOH_{14}
FFFFFFFFFFFFFFFFFFH_{15}

Figure 4. Partial product array.

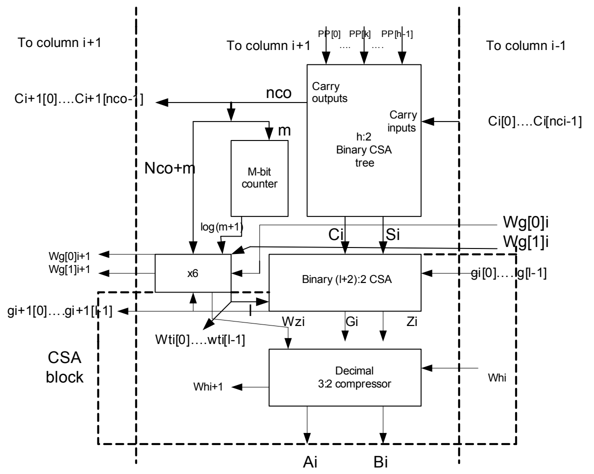


Figure 5. High level architecture of existing decimal PPR unit.

of decimal numbers and also need more number of interconnections which results in complications for implementing in FPGA. For fast FPGA implementations of BCD, carry chain addition requires nearly double the hardware than that of ripple carry addition. Since the logic depth of BCD adders also doubled, the delay of implementation will increase in proportional amount.

3.2.3 *Proposed PPR Tree with Ripple Carry Adder*

In the proposed PPR tree, the partial product reduction architecture is designed using a ripple carry adder as shown in Figure 6. The partial products generated are reduced using a tree of ripple carry adders after aligning. The number of levels used in the tree is $\log_2(2P)$ ranging from $4p + 4$-bit to $6p$-bit. The resultant of the two operand is passed to the level down adder and the ripples through the adder length. Since the carry-ripple chain overlap, signal bits are propagated as much as $8p$ bits in addition with

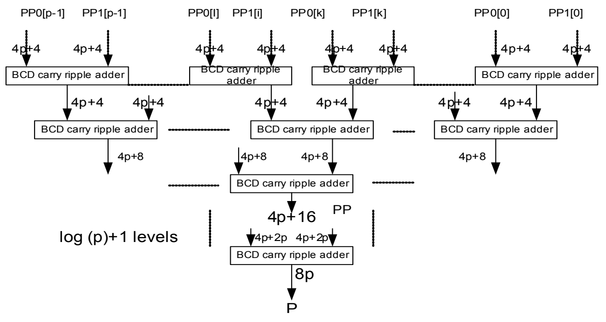


Figure 6. Proposed PPR tree for fast BCD multiplication.

$\log_2(2p)$ levels. Mathematical expression of critical path for such a design is given by

$$t_{prr} = t_{c\text{-}path} \times 2p + t_{s\text{-}path} \times (\log_2(2p) - 1) \qquad (3)$$

Where $t_{c\text{-path}}$ is delay for carry chain of 4bits (1digit) and $t_{s\text{-path}}$ is the delay for sum path of 4bits (1digit) BCD adder. Since the critical path delay only depends on number of levels and path, the required hardware resources is lowered and the delay is low, therefore speed has enhanced.

3.3 Final Conversion to BCD

The selected architecture for conversion is a 2d-digit hybrid parallel prefix or carry-select adder, the BCD quaternary tree adder. The delay of this adder is slightly higher to the delay of a binary adder of 8d-bits with a similar topology. The decimal carries are computed using a carry prefix tree, while two conditional BCD digit sums are computed out of the critical path using 4-bit digit adders which implements [Ai]+Bi+0 and [Ai]+Bi+1. These conditional sums correspond to each one of the carry input values. If the conditional carry out from a digit is one, the digit adder performs a –6 subtraction. The selection of the appropriate conditional BCD digit sums is implemented with a final level of 2:1 multiplexers. To design the carry prefix tree it is required to analyze the signal arrival profile from the PPR tree, to optimize the area for the minimum delay adder.

4. FPGA Implementation

All circuits were simulated in Verilog HDL. For most part of the multiplier it utilizes low level component instantiation. The circuits designed have been implemented on a Xilinx Virtex-5 FPGA board, device XC5VX50T 1ff1136, with a speed grade of –1. For synthesis and implementation, Xilinx ISE 14.1 tool have been used respectively. Synthesis results in terms of area, utilization of resources and delay for each N by M decimal multiplier are given in Table 2.

5. Results and Inference

In a first comparison, the proposed design matches up with three different decimal multiplication implementations on FPGAs. Table 3 shows the figures in terms of area and delay for 16x16 decimal multiplications, as well as a binary multiplier provided by Xilinx Core generation.

Table 2. Synthesis results comparison

N	M	#LUTs	#Slices	#FFs	F (MHz)	Delay (ns)
4	4	634	205	368	421.9	11.8
8	8	2109	668	1111	374.3	16
4	8	1191	380	696	392.2	15.3
8	4	1084	396	609	408.1	12.25
16	16	2868	1123	1786	327.8	24.4
16	16	2680	1012	1692	361.6	22.12

Table 3. Comparison among different architectures for 16d

Architecture	Delay(ns)	Area(# NAND)
Binary	40.8	41535
Radix-10	24.4	28925
Radix-10 (proposed)	22.12	25690

The area is measured in terms of number of NAND gates used in every logic. For example each adder is considered to have three NAND gates and each Ex-OR gate is equivalent to two NAND gates and so on. From Table 2 and Table 3 it can be inferred that, designed multiplier has lesser area (in terms of number of NAND gates) and high speed.

6. Conclusion

The combinational parallel BCD multiplier is presented in this paper with various architectural modifications. Parallel multiplication yields benefits of implementing the design in FPGA along with optimization of area and delay. The improvement of the multiplier purely depends on the use of recoding techniques like excess-3, ODDS representation and redundant codes. Which helps in quick generation of partial products, carry-free computation of precomputed terms, generating multiples in easier way just by bit inversion of positive ones. The multiplier is implemented on Virtex-5 FPGA, with a speed grade of –1. From this we can conclude that the presented work is an option for high speed implementation.

7. References

1. Vazquez A, Antelo E, Bruguera JD. Fast Radix-10 multiplication using redundant BCD codes. IEEE Transactions on Computers. 2014 Aug; 63(8):1902–14.

2. Aswal, Perumal MG, Prasanna GNS. On basic financial decimal operations on binary machines. IEEE Transactions on Computers. 2012 Aug; 61(8):1084–96.
3. Cowlishaw MF, Schwarz EM, Smith RM, Webb CF. A decimal floating-point specification. Proc 15th IEEE Symp Computer Arithmetic; 2001 Jun. p. 147–54.
4. Cowlishaw MF. Decimal floating-point: Algorism for computers. Proc 16th IEEE Symp Comput Arithmetic; 2003 Jul. p. 104–11.
5. Carlough S, Schwarz E. Power6 decimal divide. Proc 18th IEEE Symp Appl-Specific Syst, Arch, Process; 2007 Jul. p. 128–33.
6. Carlough S, Mueller S, Collura A, Kroener M. The IBM zEnterprise-196 decimal floating point accelerator. Proc 20th IEEE Symp Comput Arithmetic; Tubingen 2011 Jul 25-27. p. 139–46.
7. Dadda L. Multioperand parallel decimal adder: A mixed binary and BCD approach. IEEE Trans Comput. 2007 Oct; 56(10):1320–8.
8. Dadda L, Nannarelli A. A variant of a Radix-10 combinational multiplier. Proc IEEE Int Symp Circuits Syst; 2008 May. p. 3370–3.
9. Eisen L, Ward JW, Tast H-W, Mading N, Leenstra J, Mueller SM, Jacobi C, Preiss J, Schwarz EM, Carlough SR. IBM POWER6 accelerators: VMX and DFU. IBM J Res Dev. 2007 Nov; 51(6):p. 663–84.
10. Erle MA, Schulte MJ. Decimal multiplication via carry save addition. Proc IEEE Int Conf Appl-Specific Syst, Arch, Process; 2003 Jun 24-26. p. 348–58.
11. Erle MA, Schwarz EM, Schulte MJ. Decimal multiplication with efficient partial product generation. Proc 17th IEEE Symp Comput Arithmetic; 2005 Jun 27-29. p. 21–8.
12. Faraday Tech. Corp. 90nm UMC L90 standard performance low-K library (RVT) [Online]. 2004..
13. GorginS, Jaberipur G. A fully redundant decimal adder and its application in parallel decimal multipliers. Microelectron J. 2009 Oct; 40(10):1471–81.
14. Han L, Ko S. High speed parallel decimal multiplication with redundant internal encodings. IEEE Trans. Comput. 2013 May; 62(5):956–68.
15. Kenney RD, Schulte MJ, Erle MA. High-frequency decimal multiplier. Proc IEEE Int Conf Comput Des VLSI Comput Process; 2004 Oct 11-13. p. 26–9.
16. Kenney RD, Schulte MJ. High-speed multioperand decimal adders. IEEE Trans Comput. 2005 Aug; 54(8):953–63.