M Samatha* et al.
(IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH
Volume No.6, Issue No.5, August - September 2018, 8701-8703.

# Efficient VLSI Implementation Of Redundant Binary Encoding For Decimal Multiplication

**M.SAMATHA**
M. Tech student, Dept of ECE, Siddhartha Institute of Engineering And Technology, Hyderabad, TS, India.

**Dr. FARHA ANJUM**
Professor, Dept of ECE, Siddhartha Institute of Engineering and Technology, Hyderabad, TS, India.

*Abstract:* **This paper introduces two novel architectures for parallel decimal multipliers. Our multipliers are based on a new algorithm for decimal carry–save multioperand addition that uses a novel TCSD recoding for decimal digits. It significantly improves the area and latency of the partial product reduction tree with respect to previous proposals. We also present three schemes for fast and efficient generation of partial products in parallel. The recoding of the TCSD multiplier operand into minimally redundant signed–digit radix–10, radix–4 and radix–5 representations using new recoders reduces the complexity of partial product generation. In addition, SD radix–4 and radix–5 recodings allow the reuse of a conventional parallel binary radix–4 multiplier to perform combined binary/decimal multiplications.**

*Key words:* **Radix-10 Multiplier; Redundant Representation; Sign-Magnitude Signed Digits (Smsds); VLSI Design;**

## I. INTRODUCTION

Decimal computer arithmetic is preferred in decimal data processing environments such as scientific, commercial, financial, internet-based applications in monetary, web-based, and human interactive applications. Ever growing needs for processing power, required by applications with intensive decimal arithmetic, cannot be met by conventional slow software simulated decimal arithmetic units. However, their hardware counterparts as an integral part of recently commercialized general purpose processors are gaining importance. Binary coded decimal (BCD) encoding of decimal digits has conventionally dominated decimal arithmetic algorithms, whether realized by hardware or in software. The research for hardware realization of decimal arithmetic is not matured yet and there are rooms for improvements in hardware algorithms and designs. For example, the state-of-the-art BCD multipliers, for computing X Y, use iterative multiplication algorithms, where the partial products (i.e. the product of one BCD digit of the multiplier Y times the multi-BCD-digit multiplicand X) are generated one at a time and added to the previously accumulated result. Each partial product may be directly generated as one BCD number in [0, 9] X, or may be composed of few easy multiples of the multiplicand (e.g. 7X ¼ 4X þ 2X þ X). The latter approach tends to increase the depth (measured by the maximum number of equally weighted BCD digits) of partial product tree per each BCD digit of multiplier, which in general leads to slower partial product accumulation. But, by using possibly fast and low-cost BCD digit by BCD-digit multipliers, the former approach may lead to less costly BCD multipliers. Erle et al. have enumerated three reasons for using decimal digit-by-digit multipliers for partial product generation, which leads to less number of cycles, less wiring and no need for registers to store multiples of the multiplicand. With the rapid advances in VLSI technology, semi(fully)-parallel BCD multipliers will soon be attractive, where more than one (all) partial product(s) are generated at once and accumulated in parallel.

## II. LITERATURE SURVEY

Dynamic negation of pre computed X multiples reduces their selection cost at the penalty of one XOR gate per each bit of the selected positive multiple. This negation cost is replicated n times for parallel n×n multiplication. Moreover, the n inserted 1s for 10's complementation in and n× (n+1) 1s for digit wise two's complementation in have a negative impact on area and power saving. The same is true for the correction constant, and more complex recoding due to zero handling, for [0, 15] partial products. One way to save these costs, as we do in Section III, is to generate the SD pre computed X multiples with sign magnitude format, so as to reduce the XOR gates to one per digit (roughly 75% savings in the number of negating XOR gates) and remove the aforementioned negative impacts. However, besides slowing down the PPG to some extent (e.g., in comparison with radix-5 implementation of [6]), new problems are introduced in PPR, which are explained and solved in the next section, where we also reduce the depth of IPP matrix to n = 16, effectively prior to termination of PPG.

## III. EXISTING SYSTEM

Fast radix-10 multiplication, in particular, can be achieved via parallel partial product generation (PPG) and partial product reduction (PPR), which is, however, highly area consuming in VLSI implementations. Therefore, it is desired to lower the silicon cost, while keeping the high speed of parallel realization. Let P = X × Y represent an n × n decimal multiplication, where multiplicand X,

  

multiplier Y , and product P are normal radix-10 numbers with digits in [0, 9]. Such digits are commonly represented via binary-coded decimal (BCD) encoding. However, intermediate partial products (IPPs) are represented via a diversity of often redundant decimal digit sets. The choice of alternative IPP representations is influential on the PPG, which is of particular importance in decimal multiplication from two points of view: one is fast and low cost generation of IPPs and the other is its impact on representation of IPPs, which is influential on PPR efficiency. Straightforward PPG via BCD digit-by-digit multiplication [8], [9] is slow, expensive, and leads to n double-BCD IPPs for n×n multiplication (i.e., 2n BCD numbers to be added).

However, the work of recodes both the multiplier and multiplicand to sign magnitude signed digit (SMSD) representation and uses a more efficient 3-b by 3-b PPG. Nevertheless, following a long standing practice, most PPG schemes use pre computed multiples of multiplicand X (or X multiples). Pre computation of the complete set0, 1, . . . 9} × X, as normal BCD numbers, and the subsequent selection are also slow and costly. A common remedial technique is to use a smaller less costly set that can be achieved via fast carry-free manipulation (e.g., 0, 1, 2, 4, 5} × X) at the cost of doubling the count of BCD numbers to be added in PPR; that is, n double-BCD IPPs are generated, such as 3X = (2X, X), 7X = (5X, 2X), or 9X = (5X, 4X).

## IV. PROPOSED SYSTEM

We aim to take advantage of [−5, 5] SMSD recoding of multiplier and dynamic negation of X multiples, while reducing the number of XOR gates via generating [−6, 6] SMSD pre-computed X multiples (i.e., just one XOR gate per 4-b digit). Other contributions of this paper are highlighted below.

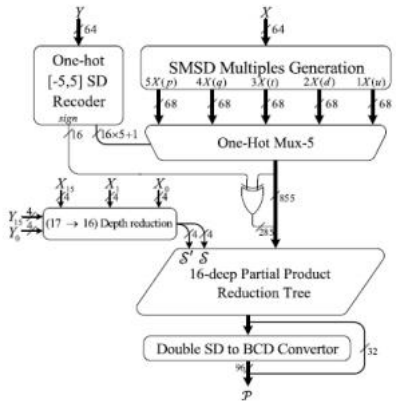


Fig. 1: Block diagram of the proposed multiplier.

1) Starting the PPR with 16 Partial Products: An especial on the fly augmentation of two middle SMSD digits leads to reducing the depth of partial product matrix by 1, such that the PPR starts with 16 operands right at the end of PPG, with no delay penalty for the latter.

2) Special 4-in-1 SMSD Adder with TCSD Sum: To avoid the challenging addition of SMSD IPPs, we design a novel carry-free adder that represents the sum of two [−6, 6] SMSD operands in [−7, 7] two's complement signed-digit (TCSD) format, where one unified adder is utilized for all the four possible sign combinations.

3) Improved TCSD Addition: The rest of the reduction process uses special TCSD adders that are actually an improved version of the fast TCSD adder. Such 2:1 reduction promotes the VLSI regularity of the PPR circuit, especially for n = 16.

4) Augmenting the Final Redundant to Non redundant Conversion with the Last PPR Level: The last PPR level would normally lead to TCSD product, which should be converted to BCD. However, to gain more speed and reduce costs, we device a special hybrid decimal adder with two TCSD inputs and a BCD output.

**Partial Product Reduction** The overall PPR for n = 16 is illustrated by Fig. 5, where a bar, triangle, square, and diamond represent a BCD, [−6, 6] SMSD, [−7, 7] TCSD, and binary signed digit (BSD), respectively. The choice of SMSD representation for the firstlevel IPPs, while facilitating the PPG, bears no extra complexity for PPR, since all reduction levels use TCSD adders, except for the first one that requires a special SMSD+SMSD-to-TCSD adder.

**Special 4-in-1 SMSD Adder:** A digit slice of the aforementioned SMSD+SMSD-toTCSD adder for four different cases corresponding to all possible combinations of the input signs is depicted by Fig. 6(a)–(d) in dot-notation representation. The black and white dots represent posibits and negabits. (A posibit is a normal bit whose arithmetic value equals its logical status, and the arithmetic value of a nega bit with logical status x equals x − 1 [24].) The sum of two [−6, 6] SMSD digits (e.g., P = sp p2 p1 p0 and Q = sqq2q1q0), and a signed carry in (e.g., Cin) is produced as one [−7, 7] TCSD digit (e.g., S = s3s2s1s0), and a signed carry out (e.g., Cout). This is a two-stage process. In the stage I, the sign bits are applied to the magnitudes, such that a negative sign changes the polarity of magnitude posibits to negabits and inverts their logical states. Subsequently, in the same stage, the bit collection U is decomposed, and the bit collection V is recoded. In the second stage, however, as will be explained shortly, only one 4-b adder takes care of all the four cases, which explains the rationale for designation of the adder.

  

To speed up the latter two steps (i.e., 2:1 reduction and TCSD-to-BCD conversion), the actual BCD product generation of Fig. 5 uses a more efficient method which is described below. The final 2:1 reduction level that is required for positions 8 to 22 and the subsequent TCSD-to-BCD conversion can be actually augmented as a TCSD + TCSD addition with BCD result.

## V. RESULTS

| Device Utilization Summary (estimated values) | | | |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of Slices | 2. | 4656 | 0% |
| Number of 4 input LUTs | 37 | 9312 | 0% |
| Number of bonded IOBs | 16 | 232 | 6% |

Figure 2:- Design summary

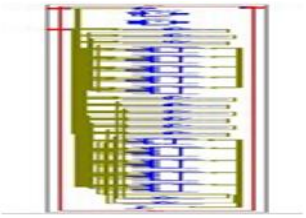

Figure 3:- RTL schematic

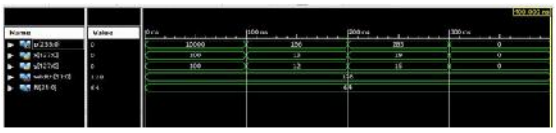

Figure 4:- Simulation results

```
Source:          x<0> (PAD)
Destination:     p<7> (PAD)

Data Path: x<0> to p<7>
                              Gate     Net
  Cell:in->out      fanout    Delay    Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   IBUF:I->O            13    1.106    0.905  x_0_IBUF (Madd_inv_x_lut<0>)
   LUT4:I1->O            3    0.612    0.520  Madd_inv_x_xor<3>11 (inv_x<3>)
   LUT3:I1->O            1    0.612    0.000  spp_0_mux0000<3>1 (spp_0_mux0000<3>1)
   MUXF5:I1->O           3    0.278    0.603  spp_0_mux0000<3>_f5 (spp_0_mux0000<3>)
   LUT4:I0 >O            1    0.612    0.509  Madd_prod_cy<5>1_SW0 (N30)
   LUT4:I0->O            3    0.612    0.520  Madd_prod_cy<5>1 (Madd_prod_cy<5>)
   LUT2:I1->O            1    0.612    0.357  Madd_prod_xor<5>11 (p_5_OBUF)
   OBUF:I->O                  3.169          p_5_OBUF (p<5>)
  ----------------------------------------
  Total                      11.028ns (7.613ns logic, 3.415ns route)
                                      (69.0% logic, 31.0% route)
```

Figure 5:- Time Summary

## VI. CONCLUSION

In this paper we have presented several techniques to implement decimal parallel multiplication in hardware. We propose three different SD encodings for the multiplier that lead to fast parallel and simple generation of partial products. For partial product reduction we have developed a decimal carry–save algorithm based on a BCD–4221 representation of decimal digit operands. It makes possible the construction ofp:2decimal CSA trees that outperform the area–delay figures of existing proposals. Moreover, proposed techniques also allow the computation of combined binary/decimal multiplications with a moderate overhead. We have proposed an architecture for decimal SD radix–10 parallel multiplication and two combined architectures for binary/decimal SD radix– 4 and binary SD radix–4/decimal SD radix–5 multiplication. The area–delay figures from a comparative study including conventional binary parallel multipliers and other representative decimal proposals show that our decimal SD radix–10 multiplier is an interesting option for high performance with moderate area.

## VII. REFERENCES

[1] IEEE standard for floating–point arithmetic. IEEE Standards Committee, Oct. 2006.

[2] F. Y. Busaba, T. Slegel, S. Carlough, C. Krygowski, and J. G. Rell. The design of the fixed point unit for the z990 microprocessor. InProc. ACM Great Lakes14th Symposium on VLSI, pages 364– 367, Apr. 2004.

[3] M. F. Cowlishaw. Decimal floating-point: Algorism for computers. InProc. IEEE16th Symposium on Computer Arithmetic, pages 104– 111, July 2003.

[4] M. A. Erle and M. J. Schulte. Decimal multiplication via carry-save addition. InProc. IEEE Int'l Conference on Application-Specific Systems, Architectures, and Processors, pages 348–358, June 2003.

[5] M. A. Erle, E. M. Schwarz, and M. J. Schulte. Decimal multiplication with efficient partial product generation. In Proc. IEEE17th Symposium on Computer Arithmetic, pages 21–28, June 2005.

[6] R. D. Kenney and M. J. Schulte. High-speed multioperand decimal adders.IEEE Trans. on Computers, 54(8):953–963, Aug. 2005.

[7] R. D. Kenney, M. J. Schulte, and M. A. Erle. High-frequency decimal multiplier. InProc. IEEE Int'l Conference on Computer Design: VLSI in Computers and Processors, pages 26–29, Oct. 2004.

[8] T. Lang and A. Nannarelli. A radix-10 combinational multiplier. InProc.40th Asilomar Conference on Signals, Systems, and Computers, pages 313–317, Oct. 2006.

[9] R. H. Larson. High-speed multiply using four input carrysave adder. IBM Tech. Disclosure Bulletin, 16(7):2053–2054, Dec. 1973.

[10] N. Ohkubo and M. Suzuki. A 4.4 ns CMOS 54x54–bit multiplier using pass-transistor multiplexer. IEEE Journal of Solid State Circuits, 30(3):251–256, Mar. 1995.

  