International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

Vlsi Implementation Of 16 × 16-Digit Parallel Multiplier

Y. Srilakshmi & T. Lilly Prasanthi
M.Tech – Scholar, Dept of E.C.E, Universal College of EngG & Tech, GUNTUR
Assistant Professor, Dept of E.C.E, Universal College of EngG & Tech, GUNTUR

ABSTRACT: *Decimal X × Y multiplication is a complex operation, where intermediate partial products (IPPs) are commonly selected from a set of pre-computed radix-10 X multiples. Some works require only [0, 5] × X via recoding digits of Y to one-hot representation of signed digits in [−5, 5]. This reduces the selection logic at the cost of one extra IPP. Two's complement signed-digit (TCSD) encoding is often used to represent IPPs, where dynamic negation (via one XOR per bit of X multiples) is required for the recoded digits of Y in [−5, −1]. In this work, despite generation of 17 IPPs, for 16-digit operands, we manage to start the partial product reduction (PPR) with 16 IPPs that enhance the VLSI regularity. Moreover, we expected to save negating XORs via representing pre-computed multiples by sign-magnitude signed-digit (SMSD) encoding. For the first-level PPR, we devise an efficient Ladner Fischer adder, with two SMSD input numbers, whose sum is represented with TCSD encoding. Expected results shows that some performance improvement over previous relevant designs.*

Keywords: Radix-10 multiplier, redundant representation, sign-magnitude signed digits (SMSDs), VLSI design

I.INTRODUCTION

Decimal arithmetic hardware is highly demanded for fast processing of decimal data in monetary, Web-based, and human interactive applications. 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. Thus, it is desired for lowering 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 generally represented through binary-coded decimal (BCD) encoding. However, intermediate partial products (IPPs) are represented through a diversity of often redundant decimal digit sets and encodings carry-save (CS) overloaded decimal [−7, 7] signed digit (SD) double 4, 2, 2, 1 and [−8, 8] SD.

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 is slow, expensive, and leads to *n* double-BCD IPPs for *n*×*n* multiplication (i.e., 2*n* 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 precomputed multiples of multiplicand *X* (or *X* multiples). Precomputation of the complete set0, 1, . . . 9} × *X*, as normal BCD

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

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 3*X* = *(*2*X*, *X)*, 7*X* = *(*5*X*, 2*X)*, or 9*X* = *(*5*X*, 4*X)*. The recoding of multiplier's digits, in some relevant works leads to a carry bit besides the *n* recoded digits of the multiplier, which will generate an extra partial product.

This is particularly problematic for parallel multiplication with *n* = 16 (i.e., number of significand's decimal digits according to IEEE standard size of single precision radix-10 floating-point numbers), where the 17 generated partial products require five PPR levels instead of four (i.e., log2 16*)*. Furthermore, they dynamically negate positive multiples based on the sign of multiplier's recoded digits. This technique reduces the area and delay of logic that selects the *X* multiples at the cost of conditionally negating the selected multiples, which requires at least 4*n*2 XOR gates for *n*×*n* multiplication.

II.EXISTED SYSTEM

The least significant product digit is obtained as an SMSD digit, which is directly converted to BCD. The next product digit that is available as a TCSD, is likewise converted. Similar is the case for TCSDs *D*3–*D*2 and *D*7–*D*4 that are delivered respectively. There are two TCSD digits. We do not apply another PPR level (i.e., TCSD+TCSD-to-TCSD conversion). Instead, we can think of a TCSD+TCSD-to-BCD converter that can be realized with the help of a parallel prefix adder.

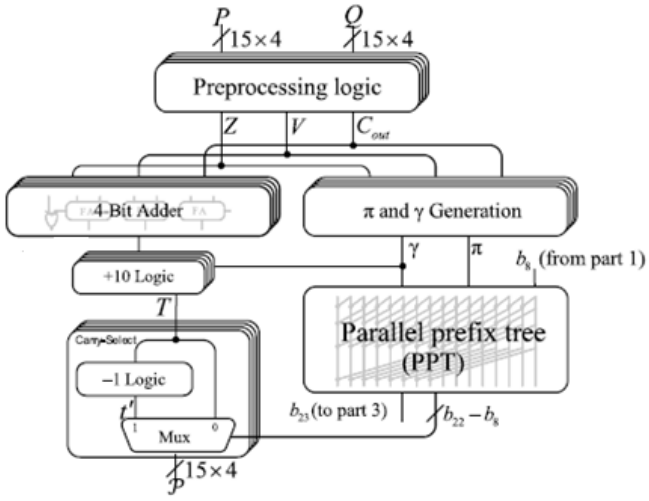

Fig 1. Existed system

A decimal borrow is carried over to the more significant decimal position that causes borrow propagation. To avoid such slow borrow propagation, we employ a parallel prefix borrow generator that uses decimal borrow propagate and generate signals *π* = *(W* = 0*)* and *γ* = *(W* < 0*)* = *w*4, respectively. These borrow signals are generated via a four level Kogge–Stone (KS) parallel prefix network with 15 input pairs (*π*, *γ)*, and borrow-in *b*8 from part 1.

To avoid 4-b borrow propagation within each digit, we also concurrently compute, where one of is to be selected by borrow *b*in that yields the product digit P. Fig. 1 depicts the logical blocks that correspond to different stages.

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

The π and γ signals for decimal positions are produced. Regarding positions where there exists only one [−7, 7] TCSD per position, γ is equal to the NOT of sign bit of the corresponding TCSD, and π can be derived as the NOR of all four bits (sign bit inverted). We devise a special three-level compound KS-like parallel prefix network to generate all borrows $b0(b-1)$ for decimal positions that correspond to the cases where $b23$ is 0 (1). Kogge stone adder depicts the required logic which represent the group (generate, propagate) signals. These borrows are utilized to form two BCD products respectively, where one is selected.

III.PROPOSED SYSTEM

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. The recoding of multiplier's digits leads to a carry bit besides the *n* recoded digits of the multiplier, which will generate an extra partial product.

In this proposed multiplier, we use Ladner-Fischer adder which is flexible to speed up the binary addition and the structure looks like tree structure for the high performance of arithmetic operations. In ripple carry adders each bit have to wait for the last bit operation. In parallel prefix adders instead of waiting for the carry propagation of the first addition, the idea here is to overlap the carry propagation of the first addition with the computation in the second addition, and so forth, since repetitive additions will be performed by a multioperand adder.

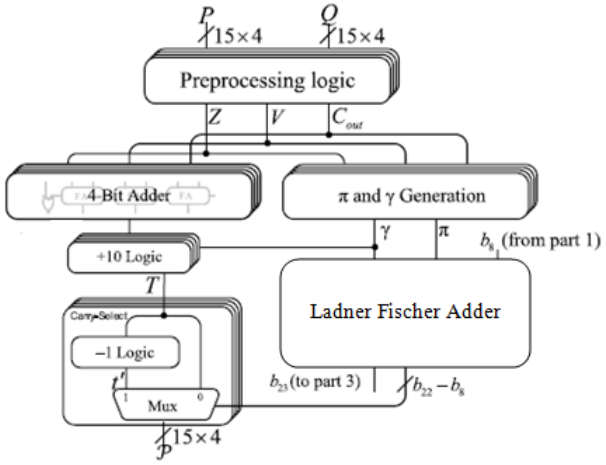

Fig 2 . Proposed system

The construction of efficient Ladner-Fischer adder consists of three stages. They are pre-processing stage, carry generation stage, post-processing stage.

A. Pre-Processing Stage: In the pre-processing stage, generate and propagate are from each pair of inputs. The propagate perform "XOR" operation of input bits and generate operation "AND" operation of input bits. The propagate (Pi) and generate (Gi) are shown in below equations 1 and 2.

$$P_i = A_i \text{ XOR } B_i \quad ----(1)$$

$$G_i = A_i \text{ AND } B_i \quad ----(2)$$

B. Carry Generation Stage: In this stage, carry is generated for each bit called as carry generate (Cg). The carry propagate and

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

carry generate is generated for the further operation but final cell present in the each bit operation gives carry. The last bit carry will help to produce sum of the next bit simultaneously till the last bit. The carry generate and carry propagate are given in below equations 3 and 4.

$$C_p = P_1 \text{ AND } P_0 - - - - - - (3)$$

$$C_g = G_1 \text{ OR } (P_1 \text{ AND } G_0) - - (4)$$

The above carry propagate Cp and carry generation Cg in equations 3 & 4 is black cell and the below shown carry generation in equation 5 is gray cell. The carry propagate is generated for the further operation but final cell present in the each bit operation gives carry. The last bit carry will help to produce sum of the next bit simultaneously till the last bit. This carry is used for the next bit sum operation, the carry generate is given in below equations 5.

$$C_g = G_1 \text{OR } (P_1 \text{ AND } G_0) - - (5)$$

C. Post-processing stage: It is the final stage of an efficient Ladner-Fischer adder, the carry of a first bit is XORed with the next bit of propagates then the output is given as sum and it is shown in equation 6.

$$S_i = P_i \text{ AND } C_{i-1} - - - - - (6)$$

It is used for two sixteen bit addition operations and each bit carry is undergoes post-processing stage with propagate, gives the final sum. The first input bits goes under pre-processing stage and it will produce propagate and generate. These propagates and generates undergoes carry generation stage produces carry generates and carry propagates, these undergoes post-processing stage and gives final sum. The step by step process of efficient Ladner-Fischer adder is shown in Fig 3.

In Efficient Ladner-Fischer adder, black cell operates three gates and gray cell operates two gates. The gray cell reduces the delay and memory because it operates only two gates. The Ladner fischer adder is design with the both black and gray cells. By using gray cell operations at the last stage of proposed adder gives a enormous dropping delay and memory used.

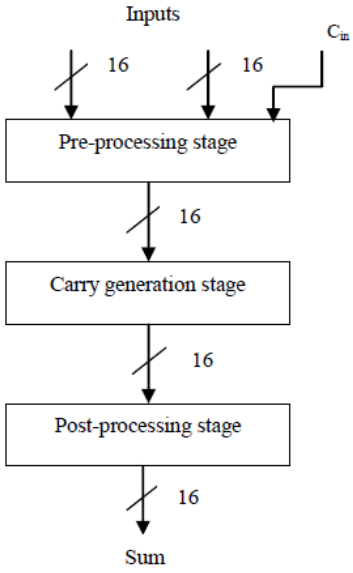

Fig 3: Flow chart for Efficient Ladner-Fischer adder.

The Ladner fischer adder is shown in fig 4 which increases the speed and decreases the memory for the operation of 8-bit addition. The input bits Ai and Bi concentrates on generate and propagate by XOR and AND operations. These propagates and generates undergoes the operations of black cell and gray cell and gives the carry Ci. That carry

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

is XORed with the propagate of next bit, that gives sum.

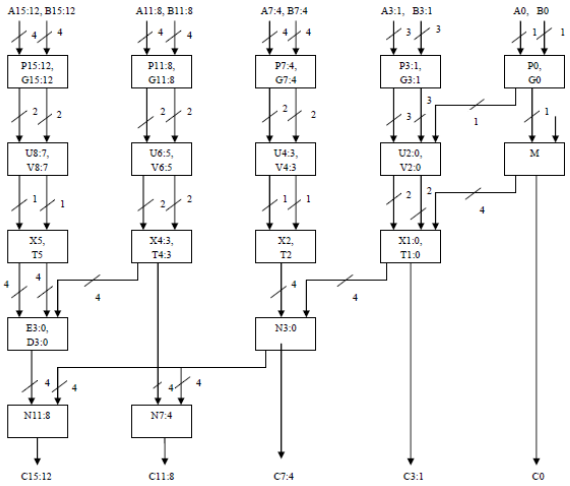

Fig 4: 16-Bit Efficient Ladner-Fischer Adder

The architecture of 16-bit Efficient Ladner-Fischer adder is shown in Fig 4. The logical circuit is using multiple adders to find the sum of N-bit numbers. Each addition operation has a carry input (Cin) which is the previous bit carry output (Cout).

IV.RESULTS

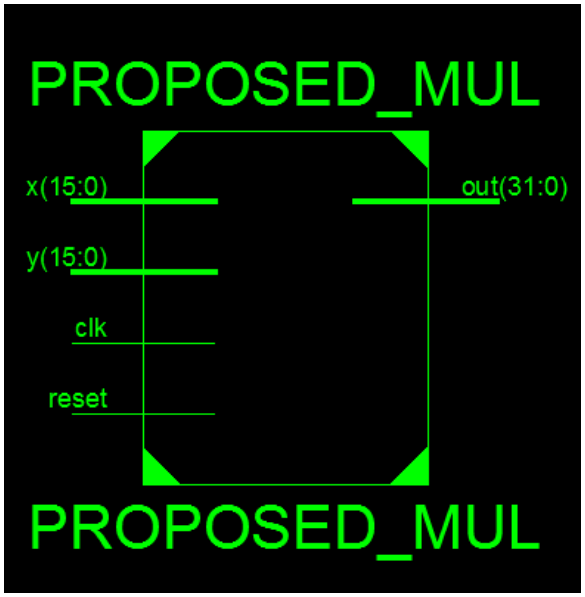

Fig 5: RTL Schematic

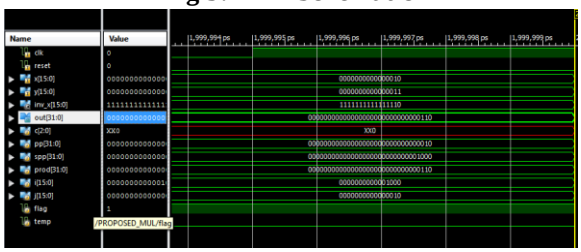

Fig 6: OUTPUT

V.CONCLUSION

We propose a parallel 16 × 16 multiplier, where 17 partial products are generated with representation. The exclusively employed representation of partial products saves more than XOR gates in comparison with other 16×16 multipliers with dynamic negation of partial products. The reason is that sign magnitude addition conceptually entails separate consideration of four sign combinations. To take advantage of early signal arrivals, conversion of the four least significant digits to starts in the middle of PPA. A Ladner Fischer adder produces sum for the nine most significant digits. A special parallel prefix decimal carry select adder adds up the middle digits and produces sum digits and a borrow that selects one of the two sums of the most significant part. The proposed muliplier occupies less area than existed multiplier.

VI.REFERENCES

[1] M. F. Cowlishaw, “Decimal floating-point: Algorism for computers,” in *Proc. 16th IEEE Symp. Comput. Arithmetic*, Jun. 2003, pp. 104–111.

[2] T. Lang and A. Nannarelli, “A radix-10 combinational multiplier,” in *Proc. 40th*

Asilomar Conf. Signals, Syst., Comput., Oct./Nov. 2006, pp. 313–317.

[3] R. D. Kenney, M. J. Schulte, and M. A. Erle, "A high-frequency decimal multiplier," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2004, pp. 26–29.

[4] A. Vazquez, E. Antelo, and J. D. Bruguera, "Fast radix-10 multiplication using redundant BCD codes," *IEEE Trans. Comput.*, vol. 63, no. 8, pp. 1902–1914, Aug. 2014.

[5] S. Gorgin and G. Jaberipur, "Fully redundant decimal arithmetic," in *Proc. 19th IEEE Symp. Comput. Arithmetic*, Jun. 2009, pp. 145–152.

[6] 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.

[7] L. Han and S.-B. Ko, "High-speed parallel decimal multiplication with redundant internal encodings," *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 956–968, May 2013, doi: 10.1109/TC.2012.35.

[8] G. Jaberipur and A. Kaivani, "Binary-coded decimal digit multipliers," *IET Comput. Digit. Techn.*, vol. 1, no. 4, pp. 377–381, 2007.

[9] R. K. James, T. K. Shahana, K. P. Jacob, and S. Sasi, "Decimal multiplication using compact BCD multiplier," in *Proc. Int. Conf. Electron. Design*, 2008, pp. 1–6.

[10] M. A. Erle, E. M. Schwarz, and M. J. Schulte, "Decimal multiplication with efficient partial product generation,"