Fast Combined Decimal/Binary Multiplier Based on Redundant BCD 4221-8421Digit Recoding

Mohammed Nabil
Electrical Engineering Department
University of Basrah
Basra, Iraq
almatajen@gmail.com

Dr. Fatemah K Al-Assfor
Computer Engineering Department
University of Basrah
Basra, Iraq
Fatimah_alasfoor@yahoo.com

Dr. Mohammed A. Al-Ebadi
Computer Engineering Department
University of Basrah
Basra, Iraq
m.a.al-ebadi@ieee.org

Abstract- **Many applications consider floating point arithmetic as a key component of the computations. Combined decimal/binary arithmetic becomes an important topic supports high speed decimal/binary applications. A new 64-bit (16×16 digit) combined decimal/binary multiplier is proposed and implemented in this work that can be used for both fused multiply add (FMA) and multiplier unit. A new partial products reduction tree is shared between decimal and binary multiplier unit. The valuation and comparison result between the proposed multiplier and the previous most recent works shows 4.66 % less delay than combined decimal/binary multiplier and 19.33 % less delay than fastest standalone decimal multiplier.**

Index Terms- Multiplier, BCD 4-2-2-1, BCD 8-4-2-1, decimal, binary.

I. Introduction

Computer hardware is mainly depending on binary arithmetic due to its hardware implementation is simpler than decimal arithmetic. However, the decimal arithmetic became a necessity in most commercial and financial applications. Thence results must be accurate and match those performed by human calculations. Because of the importance of the decimal, IEEE submits decimal floating-point specification in IEEE 754-2008 standard [1]. The main concern of the decimal numbers is that they cannot be represented as a binary number with a finite number of bits, the further 4-bit combinations, i.e. 1010 to 1111, are not valid in decimal system [2]. The simplest multiplication process is to directly determine the product of two operands $P = N1 \times N2$, where *N1* represents the multiplicand, *N2* the multiplier, (*PP*) the partial products and (*P*) represents the product. The two vectors (sum and carry) are located in the last two rows of the multiplication operation sequence before the product. All the bits of the *PPs* in each column are reduced to obtain two vectors: sum and carry [3]. Decimal and binary multiplication can be implemented in parallel approaches. A parallel Multiplier approach is divided into five main stages which are multiplier operand recoding, multiplicand multiples generation, generation of partial product (*PP*), reduction of *PP* and carry propagation addition methods to calculate the redundant final result [4].

Decimal multiplier usually has larger area and longer delay than its binary counterpart; thus, architecture which supports both decimal and binary arithmetic unit should be shared some hardware components. A variety of redundant BCD codes like 4221, 5211, and 3321 might be used in decimal arithmetic as the sum of 4-bit equal to 9 that satisfy the decimal number representation (0,1,2,…… 9). The BCD 4-2-2-1 and BCD 5-2-1-1 codes are very ordinary because they supported 4 bits carry save addition (CSA) and module X2 operation [5].

In this paper, a combined multiplier is proposed to speed up performance of the floating point unit. Several techniques have been proposed to enhance the multiplier performance. These techniques are: 1) a new *PPR* tree is proposed that uses column by column accumulation to add the *PPs* into two vectors (sum and carry); 2) (9:4) and (8:4) compressors based on bit counters uses to simplify the reduction stage; 3) a pre-alignment approach with multiplier array is suggested; 4) as long as digits are represented in BCD format, there is no need for additional correction hardware circuit other than module X2 in decimal and 1-bit left shift (L1) operation in binary [6-7].

The rest of the paper consists of six sections; section II reviews related works. Proposed deign is described in section III. The simulation results are depicted in section IV. Section V determines the delay and compares the result with the recent related works. Finally, section VI shows the conclusions.

II. RELATED WORKS

Recently, several hardware implementations of digital multiplier have been adopted for combined decimal and binary and others for standalone parallel decimal multiplier. Hickmann et al. [8] have presented a combined decimal and binary multiplier design with low-latency and an efficient area by replacing the (4:2) CSA with (3:2) CSA in the partial products tree to decrease the number of the decimal doubling units and propose a split decimal/binary multiplier tree. Vestias et al. [9] had proposed (8×8) and (16×16) decimal multiplier using binary multiplication method. L.Han et al. [10] have presented a new technique of parallel decimal multiplication using redundant digit set [-8,8] approach. Also, (*P+1*) sign digit (SD) partial products have been generated which can be reduced using new multi operand SD addition. Zhu et al. [11] have presented decimal multiplier using the advantage of the BCD 8-4-2-1 and BCD 5-4-2-1 recoding to improve the pre-computations of (2A) and (5A). Also the work is introduced a new BCD 4-2-2-1multiplier based on a navel BCD 4-2-2-1 full adder. Cui et al.[12] have proposed a new partial product tree. The carries which are generated between the digit columns had been counted using a BCD 4-2-2-1 sum correction circuit. Also decimal (3:2) compressors are used in the PPR tree to reduce the result of 6*x carries count and binary CSA PPR tree. Wahaba et al.[13] have presented a combined decimal and binary multiplier based on multi-operand adder using decimal to binary conversion approach. From the related works, it is clear that the BCD 4-2-2-1 format has three interesting properties. First, all the 16 possible BCD 4-2-2-1 values had valid representation in decimal format. Second, BCD 4-2-2-1 supports a decimal doubling unit which is required only two levels of logic function. Third, BCD 4-2-2-1 is a self-complement which supports the (9's) complement.

Table I illustrates the BCD 8-4-2-1, BCD 4-2-2-1, and BCD 5-2-1-1 recoding which are used in this work.

1. PROPOSED COMBINED DECIMAL/BINARY MULTIPLIER

2. The aim of this work is to design and implement an efficient multiplier to be utilized in both decimal and binary units based on the parallel multiplication approach with sharing some hardware resources. Most of the multipliers are partitioned into the same main blocks, but the difference of them is in the implementation way of each block and how they are combined together to achieve a high performance and reduced the path delay. The decimal multipliers described in [7] are reconfigured and improved in this work. These improvements let the CSA reduction tree compatible between decimal and binary path. The partial products tree is shared between the decimal and binary because of the 1-bit left shift (L1), which is explained in Fig.1, is used in binary partial product reduction stage for correction purpose.

A binary operand is recoded to binary SD Radix-4 while a decimal operand is recoded to decimal SD Radix-5 that has lower latency in multiplicand multiples generation stage. This is because of SD Radix-4 requires additional partial products that increases the area during the partial product selection stage, and both upper and lower partial product digits $(PP^U[i]$ and $PP^L[i])$ resultant in BCD 4221 format.

The top level of the proposed combined multiplier is illustrated in Fig.2. The selection and reduction of the partial product stages are shared between the decimal and binary operation while the generation of multiplicand multiple and the recoding of multipliers are separated for each data type. The idea behind decimal multiplies recoded into Radix-5 to generate 32-digit that is the same number of partial products of the binary multiplication. A set of 2-to-1 MUXs are inserted after the multiplicand multiples generation and multiplier recoding stages to determine the current operation type either decimal or binary multiplication. According to decimal SD Radix-5 and binary SD Radix-4, $(2 \times P)$ partial products for the decimal and binary multiplication will be generated. Furthermore, an additional PP of 1X (which means no shift) is required in the last row of the reduction tree to correct the result when the last recoded sign digit (SD) is negative during the multiplication of 64-bit unsigned binary number. The total numbers of partial products are equal {($2 \times P$) +1}.

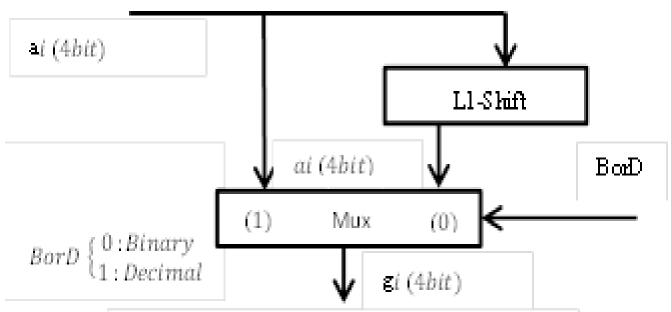


Fig. 1 1-bit left shift (L1)

Table I
BCD 8-4-2-1, BCD 4-2-2-1 and BCD 5-2-1-1 Recoding

Decimal	BCD 8-4-2-1	BCD 4-2-2-1	BCD 5-2-1-1
0	0000	0000	0000
1	0001	0001	0010 or 0001
2	0010	0010 or 0100	0100 or 0011
3	0011	0011 or 0101	0101 or 0110
4	0100	1000 or 0110	0111
5	0101	1001 or 0111	1000
6	0110	1010 or 1100	1010 or 1001
7	0111	1011 or 1101	1100 or 1011
8	1000	1110	1110 or 1101
9	1001	1111	1111

A. *Multiplier Operand Recoding techniques*

The advantage of using decimal SD Radix-5 and binary SD Radix-4 together in the proposed multiplier is that each multiplier digit is divided to two parts (upper and lower) to generate the two PPs.

In SD Radix-4 recoding, each 4-bit of multiplier $B_i \in$ (0, 1, 2, 3,…, 15), the carry-in (C_{in}) bit from the lower 4-bit, and the carry out (C_{out}) bit to the next four bits are utilized to generate two digits: lower and upper, as illustrated in Eq. (1).

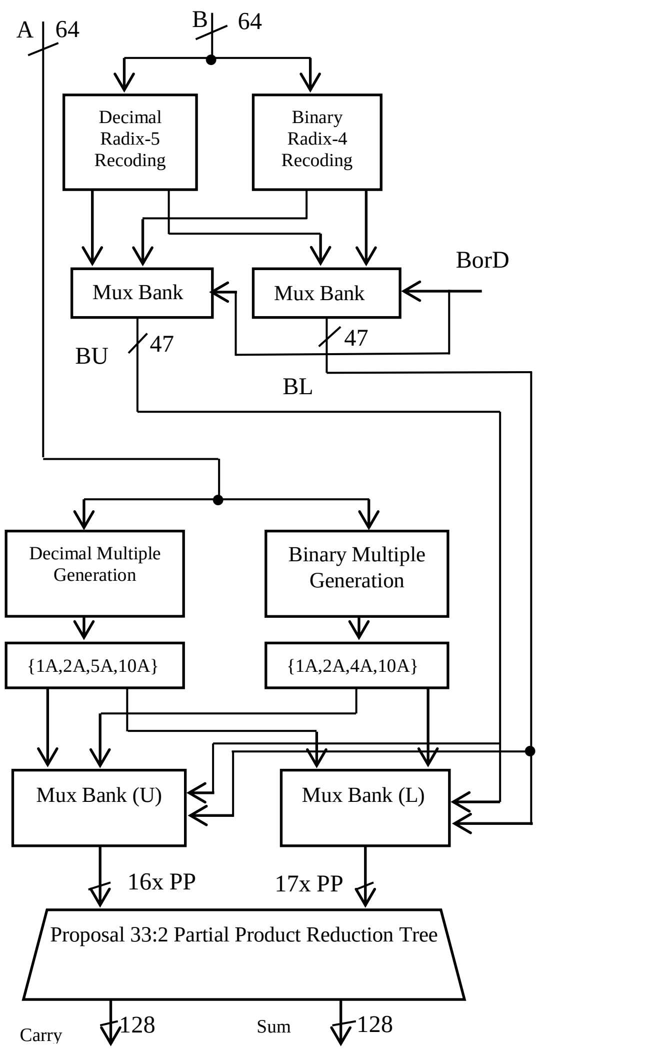


Fig. 2 Top Level of the Combined Multiplier

$$B_i + C_{in} = 16 \times C_{out} + 4 \times B_i^U + B_i^L \qquad (1)$$

where $B_i^U \in \{-2, -1, 0, 1, 2\}$ represents the upper digit and $B_i^L \in \{-2, -1, 0, 1, 2\}$ represents the lower digit.

A carry out (C_{out}) bit is generated from the input digit and it doesn't count on the value of carry in (C_{in}) bit. Furthermore, it does not add any carry propagation delay overhead.

The selection signals of SD Radix-4 can be generated from the binary input operand (B) per digit by Eq.(2) and (3):

$$(B_i^L) \left\{ \begin{array}{l} b(+1)_i^L = (\overline{b_{i,1}} . \overline{c_{in}} . b_{i,0}) + (\overline{b_{i,1}} . c_{in} . \overline{b_{i,0}}); \\ b(+2)_i^L = \overline{b_{i,1}} . b_{i,0} . c_{in}; \\ b(-1)_i^L = (\overline{b_{i,0}}\ c_{in} . b_{i,1} .) + (b_{i,0} . \overline{c_{in}} . b_{i,1}); \\ b(-2)_i^L = b_{i,1} . \overline{b_{i,0}} . \overline{c_{in}}; \\ bs_i^L = (b_{i,1} . \overline{b_{i,0}}) + (b_{i,1} . \overline{c_{in}}); \end{array} \right. \qquad (2)$$

$$(B_i^U) \left\{ \begin{array}{l} b(+4)_i^U = (\overline{b_{i,3}} . \overline{b_{i,2}} . b_{i,1}) + (b_{i,2} . \overline{b_{i,1}} . \overline{b_{i,3}}); \\ b(+8)_i^U = b_{i,1} . b_{i,2} . \overline{b_{i,3}}; \\ b(-4)_i^U = (\overline{b_{i,2}} . b_{i,1} . b_{i,3}) + (\overline{b_{i,1}} . b_{i,2} . b_{i,3}); \\ b(-8)_i^U = b_{i,3} . \overline{b_{i,2}} . \overline{b_{i,1}}; \\ bs_i^U = (\overline{b_{i,2}} . b_{i,3}) + (b_{i,3} . \overline{b_{i,1}}); \\ Cout_i = b_{i,3} \end{array} \right. \qquad (3)$$

For decimal SD Radix-5 recoding, each BCD 4-2-2-1 digit of the Multiple B ∈ (0, 1, 2, 3… 9) is recoded into two digits (B_i^U and B_i^L) that represents an upper and lower digit in addition to the sign bit. This recoding has different sets of multiplicand multiplies (5A, 10A instead of 4A, 8A) which are generated by various logical expressions. The multiplicand multiplies {0, A, 2A} can be selected by lower digit while the upper digit is used to select {0, 5A, 10A} according to Eq. (4)

$$B_i = B_i^U \times 5 + B_i^L \qquad (4)$$

Where $B_i^U \in \{2, 1, 0\}$ and $B_i^L \in \{2, 1, 0, -1, -2\}$. The selection signals of SD Radix-5 can be generated from multiplier digit (B) using Eq. (5):-

$$(B_i^U) \left\{ \begin{array}{l} b2_i^U = b_{i,3}; \\ b1_i^U = b_{i,2} + (b_{i,1} . b_{i,0}); \end{array} \right.$$

$$(B_i^L) \left\{ \begin{array}{l} b(+2)_i^L = b_{i,1} . ((b_{i,0} . b_{i,2}) + (\overline{b_{i,0}} . \overline{b_{i,2}})); \\ b(+1)_i^L = (\overline{b_{i,1}} . b_{i,0} . \overline{b_{i,2}} . \overline{b_{i,3}}) + (b_{i,1} . b_{i,2} . \overline{b_{i,0}}) \\ b(-1)_i^L = (b_{i,3} . b_{i,0}) + (\overline{b_{i,1}} . b_{i,2} . \overline{b_{i,0}}); \\ b(-2)_i^L = (b_{i,3} . \overline{b_{i,0}}) + (b_{i,0} . \overline{b_{i,2}} . b_{i,1}); \\ bs_i = b(-2)_i^L + b(-1)_i^L; \end{array} \right.$$

Where $A_{i,j}$ and $w_{i,j}$ are the operand represented in BCD 4-2-2-1 and BCD 8-4-2-1, respectively.

The selection of partial products stage outputs 32 partial products, some of them are negative and need their sign to be extended. The detail structure of the multiplier digit recoder and multiplicand multiples selection are illustrated in Fig. 3.

Generation of Multiplicand Multiples

In this stage, a set of multiplicand multiples had been generated in parallel according to each multiplier digit. Binary radix-4 ∈ ($\pm A$, $\pm 2A$, $\pm 4A$ and $\pm 8A$) multiplies are coded to BCD 8-4-2-1 and its counterpart decimal radix-5 ∈ ($\pm A$, $\pm 2A$, $\pm 5A$ and $\pm 10A$) multiplies are coded to BCD 4-2-2-1.

The multiples of binary multiplicand are obtained directly by m-bit left shifts (Lm_{shift}) where m ∈ (1, 2, 3, 4). Negative multiple is performed by bitwise inversion (one's complement).

The decimal multiplicand multiplies A, 2A, 5A and 10A are generated in simple levels of logical expression using an recoding methods and left shift. It can be generated as follows:

Multiple A: the BCD 8-4-2-1 multiplicand is recoded to BCD 4-2-2-1 easily as follow:

$$(w_{i,0}, w_{i,1}, w_{i,2}, w_{i,3}) = (A_{i,0}, A_{i,3} \curlyvee A_{i,1}, A_{i,3}, A_{i,3} \curlyvee A_{i,2})$$
(6)

Multiple 2A: each BCD 8-4-2-1 is recoded into BCD 5-2-1-1 using Eq. (7):

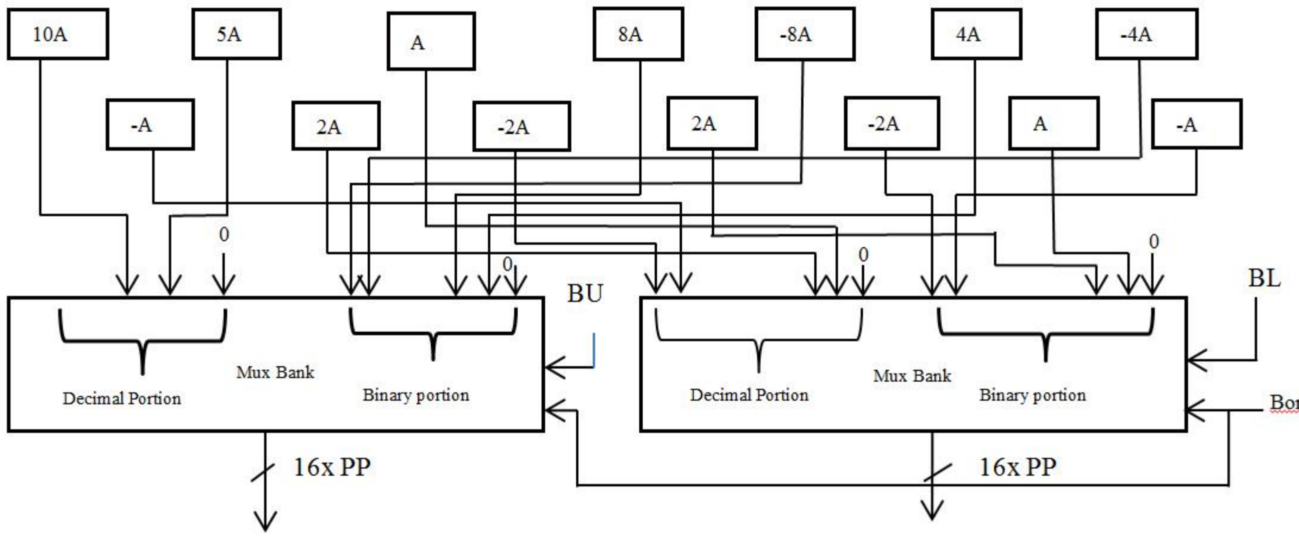


Fig.3 Partial Product Generation for Combined Binary Radix-4/Decimal Radix-5

$$w_{i,3} = a_{i,3} \curlyvee a_{i,2} . (a_{i,0} \curlyvee a_{i,1});$$
$$w_{i,2} = a_{i,3} \curlyvee ((a_{i,2} . \overline{a_{i,0}}) \oplus a_{i,1});$$
$$w_{i,1} = a_{i,3} . a_{i,0} \curlyvee a_{i,2} . (\overline{a_{i,1} \curlyvee a_{i,0}});$$
$$w_{i,0} = a_{i,3} \curlyvee (a_{i,2} \oplus a_{i,0});$$

Then, 1-bit left shift (L1) operation is done on the recoded multiplicand result to obtain 2A multiple results in BCD 4-2-2-1.

Multiple 5A: First, each BCD 8-4-2-1 is shifted 3-bit to the left by considering the result in BCD 5-4-2-1.Then; this result is recoded from BCD 5-4-2-1 into BCD 4-2-2-1 according to Eq. (8):

$$\left.\begin{array}{l} w_{i,3} = a_{i,2} \curlyvee a_{i,3}\ ; \\ w_{i,2} = a_{i,3} \cdot (a_{i,2} \curlyvee (a_{i,0} \cdot a_{i,1}))\ ; \\ w_{i,1} = a_{i,1} \cdot a_{i,3} \cdot (a_{i,0} \curlyvee a_{i,2});\\ w_{i,0} = a_{i,3} \oplus a_{i,0}\ ; \end{array}\right\} \qquad (8)$$

Multiple 10A: each BCD 8-4-2-1 is recoded into BCD 4-2-2-1 using Eq. (6) then the result is shifted 4-bit to left.

The negative multiples generation are formed according to 10's complement of the positive multiples, as below:

$$-A = \sum_{i=0}^{d-1} (9 - A_i) \cdot 10^i + 1 \qquad (9)$$

Two MUXs are used to select two appropriate multiplicand multiples according to each multiplier digit.

B. *Reduction of Partial Product*

Both BCD 4-2-2-1 and BCD 8-4-2-1 formats are used in the partial products reduction stage. This stage consists of: (9:4) and (8:4) compressors, (3:2) CSA tree to reduce each column $(p:2)$ into a two vectors (sum and carry), module X2, and 1-bit left shift (L1) operations.

1) Decimal/Binary Carry Free Adders (CFA)

The proposed multiplier uses two types of compressors: (9:4) and (8:4) to add the input partial products based on bit counter methods for decimal and binary operands coded in BCD 4-2-2-1 and BCD 8-4-2-1. These compressors composed of a row of bit counters which reduce a column of 9 or 8 bits into a 4-bit vector coded in BCD 4-2-2-1 or BCD 8-4-2-1 for decimal or binary, respectively. Fig. 4 shows the implementation of the (9:4) compressor which can reduce up to 9 bits using two levels of full adder. The output of the binary weight is indicated by a bracket. The path delay of (9:4) and (8:4) compressors are equal to the path delay of binary (4:2) CSA. To illustrate the algorithm, Fig.5 shows an example of reduction an 8-decimal digit coded in BCD 4-2-2-1 and the result is coded in BCD 4-2-2-1 too.

The 4-bit of each digit is take place in a column. From top represents the most significant bit to the bottom represents least significant bit and aligned in a 4-row.

2) Decimal/Binary (3:2) Carry Save Adder

The conventional 4-bit (3:2) CSA is used in the second stage of the proposed reduction tree to add the output result from the first level of (9:4) and (8:4) compressors.

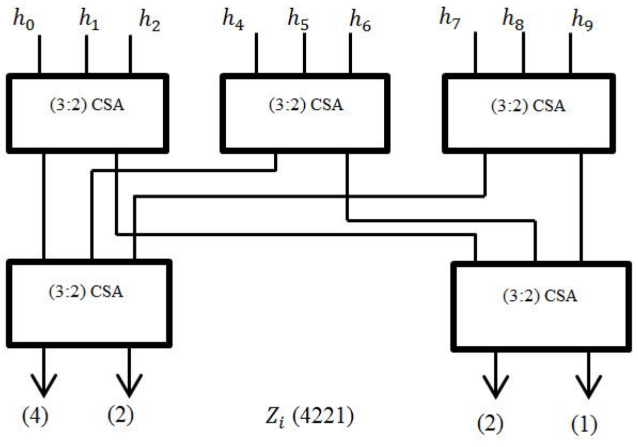


Fig.4 Implementation of (9:4) Compressor

Decimal (3:2) CSA consists of a binary (3:2) CSA and module X2. The module X2 operation, shown in Fig.6, consists of digit conversion from BCD 4-2-2-1 to BCD 5-2-1-1, then the output is shifted to the left by 1-bit. A 4-bit MUX was controlled by BorD control signal to determine whether the CSA is decimal or binary.

Figure.7 explains the decimal/binary CSA(s) reduction tree. The X4 block can be obtained by two cascaded of modules X2.

3) Combined Decimal/Binary CSA Reduction Tree

A new partial products tree is used to reduce $(p$:2) rows where (2 $\leq p \leq$ 33) of Z_i[k] input digits with weight 10^k or input 4-bit Z_i [k] with weight 2^i into two vectors S_i and H_i (4 bits for each one).The input digits Z_i is coded either in BCD 4-2-2-1 for decimal case or in BCD 8-4-2-1 for binary case. The idea of the CSA tree is to use the tree for both decimal and binary computations.

In case of decimal, the input operands are recoded in BCD 4-2-2-1. Whereas; in case of binary, the input operands are represented by BCD 8-4-2-1 digits each of 4-bit.

As presented in Fig.7, the proposed partial product tree consists of three parts:

1. For$(0 \leq p \leq 33)$, the input digits Z_i [k] reduced in the first level using the (9:4) and (8:4) compressors. The output digits are multiplied by 2 or 4 (i.e. X2 or X4) and it is corrected by L1 block, where L1 (1-bit left shift). Then, the output from the first level is being reduced by the second level, which includes a set of regular binary 4-bit CSA (3:2) tree. However, each module X2 or L1 operation produces C_{out} bit to the upcoming significant digit column of the *PPs* array.
2. A new pre-alignment for *PP* array has been proposed as described in Fig.8. The numbers of the generated partial product equal to *2d+1* where *d* is 64-bit (16-digit) input operand. Both PP^L[i] (4221/8421) and PP^U[i] (4221/8421) have the same weight $10^i/2^i$ in terms of decimal/binary multiplication, respectively**.** The 33 partial products resulted are aligned in an array.
3. As shown in Fig.8, the partial product array comprise of 33 columns. The numbers of rows in the array are equal to the number of digits per each column. Each column is reduced independently using the CSA reduction tree.

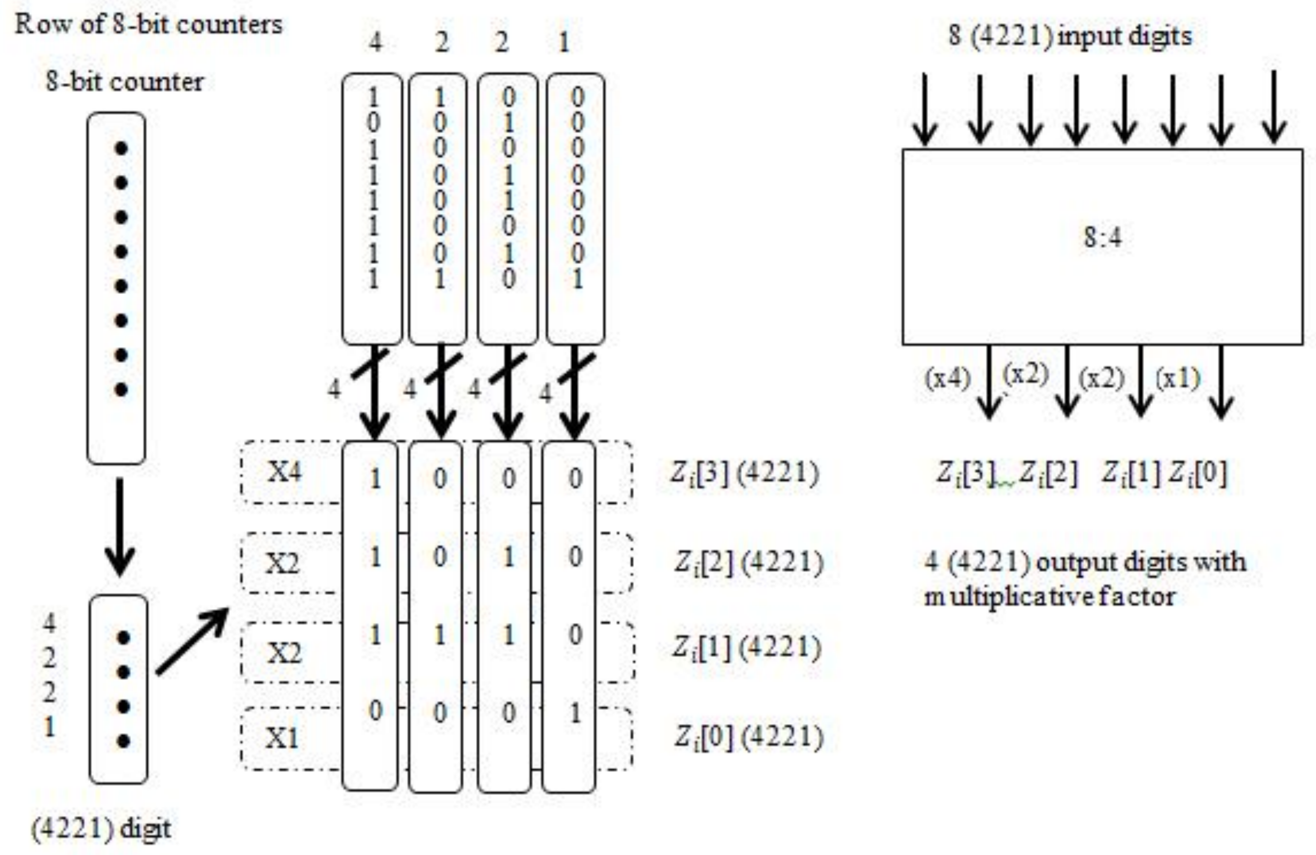


Fig. 5 (8:4) Reduction of (4221) Decimal Code Operands

To balance the overall delay, the slowest output of each (3:2) CSA stage is connected directly to the fast input of the next (3:2) carry save adder stage.

It can be shown from Fig.8 that the worst case of reduction in the *PPs* tree is in column no.16 where the total number of digits is equal 33.

III. THE SIMULATION RESULT

The proposed combined multiplier 64-bit (16x16 digits) is implemented at gate level, then the result is compared with the digital multipliers presents in [12] [13] using logical effort approach. The proposed multiplier is synthesized with Xilinx ISE13.2.

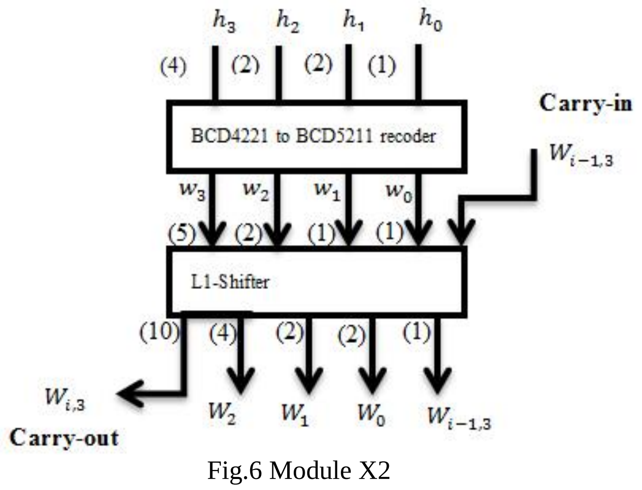


Fig.6 Module X2

B. Numerical Example

For the decimal case, the multiplication of two numbers N1_d and N2_d are given below:

Let N1_d = 7555255216 and N2_d = 7583916655 and the Products of (N1_d × N2_d) = 5729 8425 8653 9802 2480

Fig. 9 shows the simulation result of the decimal multiplication.

For the binary case, the simulation result of the following example of multiplying two numbers A and B is illustrated in Fig.10

N1_b= $355E31CE5_{16}$

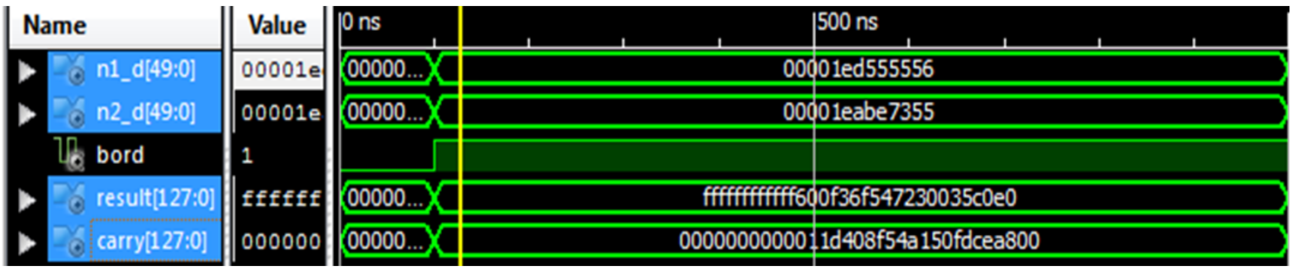
N2_b= $35794A54A_{16}$

(N1_b × N2_b)=$25C9541fBCA9F332_{16}$

II. COMPARISON

The logical effort approach is a way used to measure the delay in unit of FO4 in a CMOS circuit. The fastest design can be selected by comparing the delay of logic structures in the different component types.

Logical effort approach is used to calculate the estimation delay of the proposed combined multiplier. The estimated delay of the proposed combined multiplier is compared with the most recent combined decimal/binary multiplier [13] and the standalone decimal multiplier [12].



```
Name           Value    0 ns                      500 ns
n1_d[49:0]     00001e   00000...  00001ed555556
n2_d[49:0]     00001e   00000...  00001eabe7355
bord           1
result[127:0]  ffffff   00000...  ffffffffffff600f36f547230035c0e0
carry[127:0]   000000   00000...  000000000001d408f54a150fdcea800
```

Fig.7 Proposed Decimal/Binary CSA(s) Reduction Tree

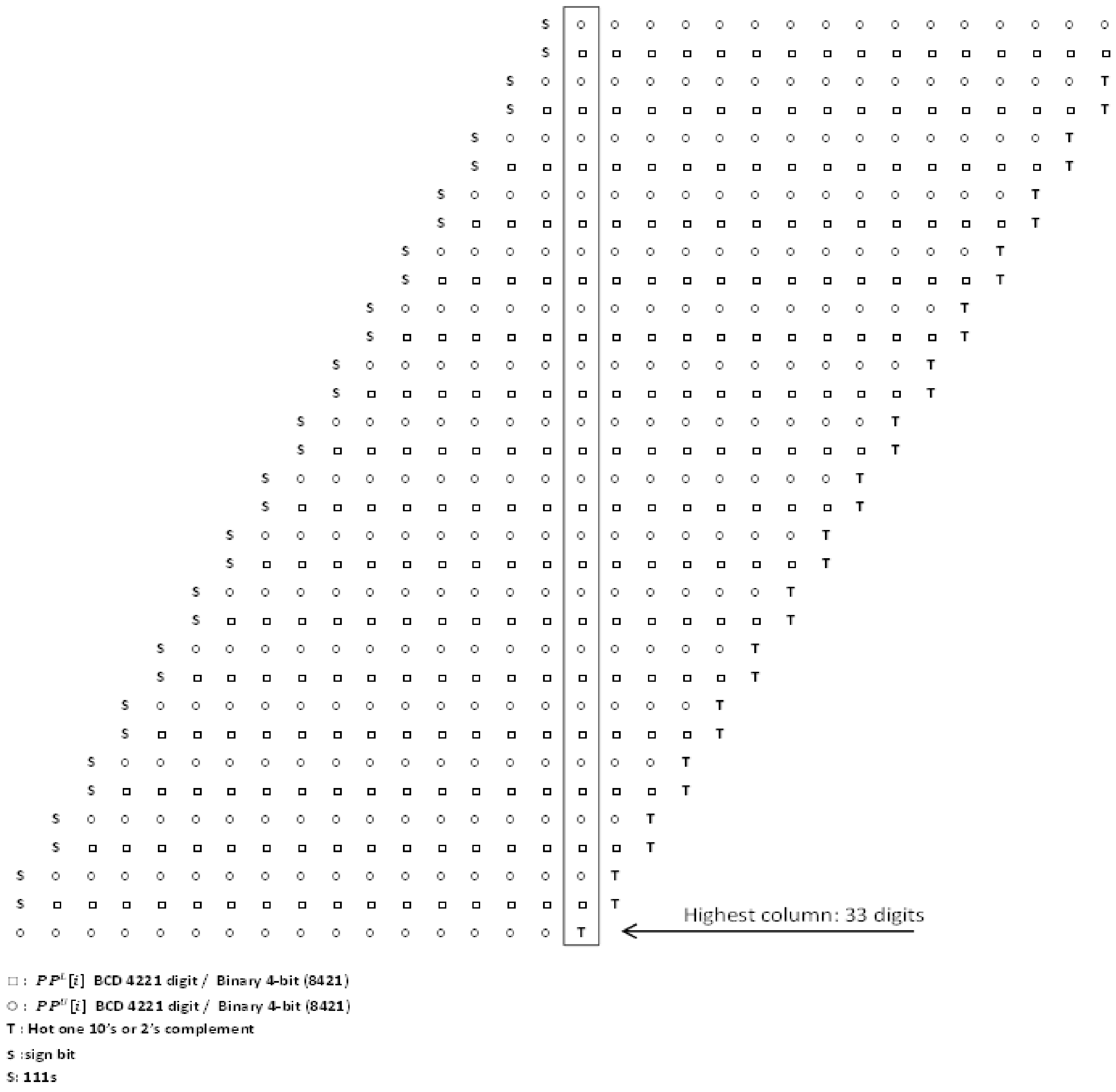

Highest column: 33 digits
□ : $PP^{L}[i]$ BCD 4221 digit / Binary 4-bit (8421)
○ : $PP^{U}[i]$ BCD 4221 digit / Binary 4-bit (8421)
T : Hot one 10's or 2's complement
s :sign bit
S: 111s

Fig. 8 Partial Product Reduction Alignments

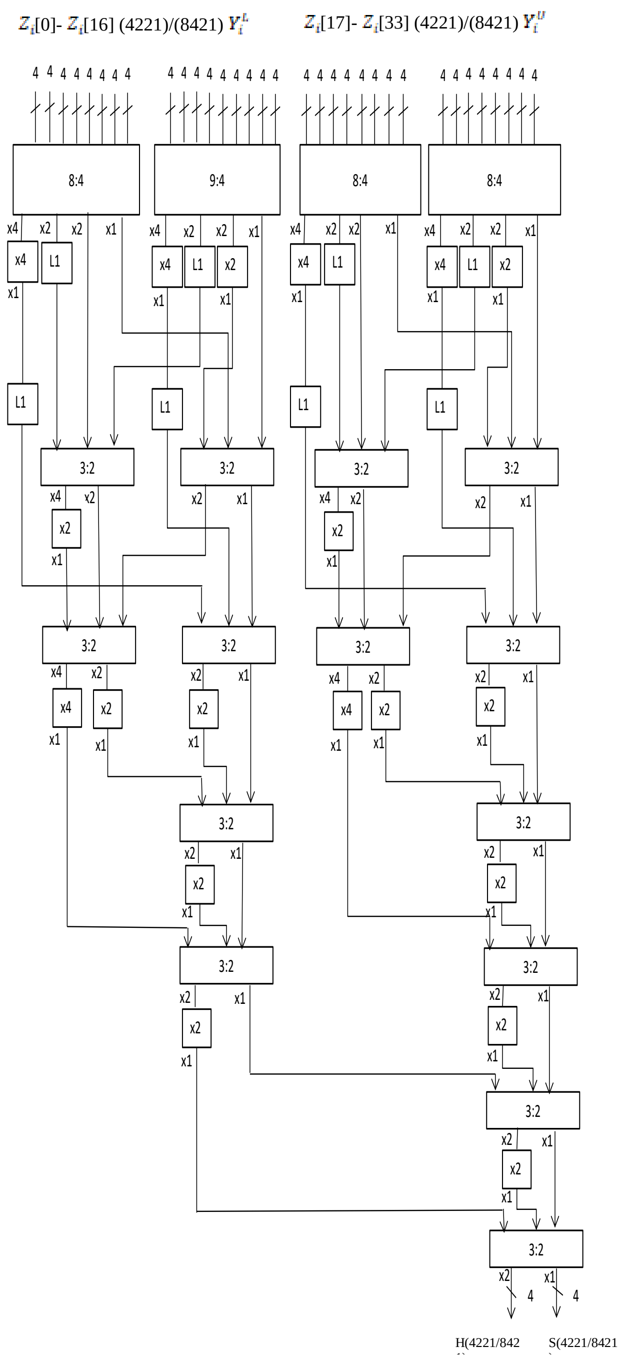

Z_i[0]- Z_i[16] (4221)/(8421) Y_i^L
Z_i[17]- Z_i[33] (4221)/(8421) Y_i^U
8:4
9:4
8:4
8:4
3:2
x4
x2
x1
L1
H(4221/842
S(4221/8421

Fig. 9 Simulation Result of Decimal Multiplication

Table II shows the delay calculation of the multiplier.

Table II
Delay Calculation of Proposed Multiplier Architecture (FO4)

Block	Delay (FO4)
PP Generation stage	6.1
PP Reduction tree	31.3
Adder stage	3.2
Total stages	**40.5**

Table III illustrates that the proposed combined multiplier reduces the delay by 4.66% and 19.32% when compared with [13] and [12], respectively.

Table III
Delay Comparison between Different Multipliers

Design	Delay (FO4)
Decimal Multiplier [12]	50.2
Combined Multiplier [11]	42.48
Proposed Multiplier	**40.5**

Note: a) Binary PPR tree only = 20.8 FO4.
b) Decimal PPR tree only = 26.5 FO4.

III. Conclusions

In this research, a new combined decimal/binary multiplier based on a new partial products reduction tree has been proposed. It supports both decimal and binary floating point units. A new CSA tree based on (9:4) and (8:4) compressors is suggested and used to reduce the partial products column by column. Then, a set of conventional (3:2) CSA is used to add the result of the (9:4) and (8:4) compressors. The proposed fast combined multiplier can be used in both high performance multiplier and fused multiply add (FMA) units.

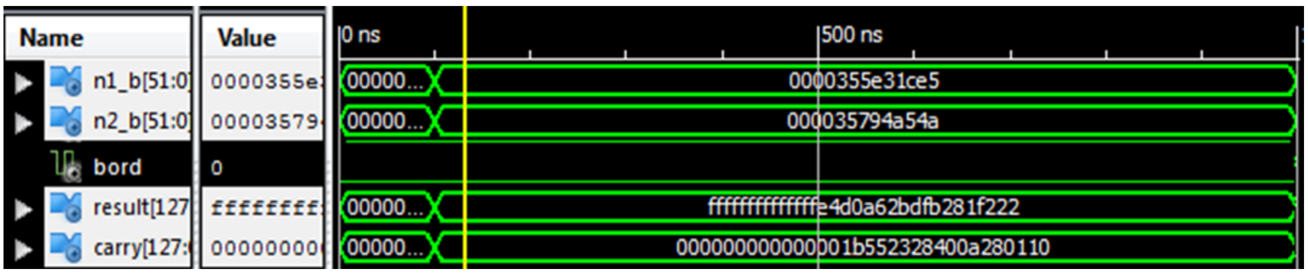

```
Name          Value      0 ns                    500 ns
n1_b[51:0]    0000355e   00000...  0000355e31ce5
n2_b[51:0]    00003579   00000...  000035794a54a
bord          0
result[127    ffffffff   00000...  ffffffffffffffe4d0a62bdfb281f222
carry[127:    00000000   00000...  00000000000001b552328400a280110
```

Fig. 10 Simulation Result of Binary Multiplication

IV. References

[1] D. Zuras and M. Cowlishaw, "IEEE Standard for Floating Point Arithmetic", IEEE 754-2008, pp. 1–70, Aug. 2008.

[2] T. Lang and A. Nannarelli, "A Radix 10 Combinational Multiplier", In 40-th Asilomar Conference on Systems computers and signals, pp. 313-317, Oct.2006.

[3] A. Vazquez , J. Bruguera and E. Antelo, " Fast Radix-10 Multiplication Using Redundant BCD codes", In IEEE Transactions on Computers, Vol. 63, No. 8, pp. 1902-1914, Aug. 2014.

[4] L. Dadda and A. Nannarelli, "A Variant of a Radix-10 Combinational Multiplier", In IEEE Systems and Circuits (ISCAS), pp 3370-3373, May. 2008.

[5] M. Erle, E. M. Schwarz and M. Schulte, "Decimal Multiplication With Efficient Partial Product Generation", In 17-th IEEE Symposium on Computer Arithmetic, pp 21-28, Jun. 2005.

[6] A. Vazquez ,P. Montuschi and E. Antelo, "A New Family of High-Performance Parallel Decimal Multipliers", In. 18-th IEEE Symposium on Computer Arithmetic, pp 195-204, Jun. 2007.

[7] A. Vazquez, P. Montuschi and E. Antelo, "Improved Design of High Performance Parallel Decimal Multipliers", IEEE Transactions on Computers, Vol. 9, No. 5, pp 679-693, May. 2010.

[8] B. Hickmann and Mark Erle, "Improved Combined Decimal/Binary Fixed Point Multiplier", IEEE, pp 87-94, Aug. 2008.

[9] M. Vestias and H. Neto, "Parallel decimal multipliers using binary multipliers", IEEE Programmable Logic Conference (SPL), pp.73-78, Jun.2010.

[10] L. Han. and S. Ko, "High-Speed Parallel Decimal Multiplication with Redundant Internal Encodings", IEEE Transactions on Computers, Vol.62, pp 956–968, Jan.2012.

[11] M. Zhu , A. Baker and Y. Jiang, "On a parallel decimal multiplier based on hybrid 8-4-2-1 and 5-4--21 BCD recoding", In. IEEE 56-th International Midwest Symposium, pp. 1391-1394, Aug.2013.

[12] X. Cui,W. Liu,D. Wenwen and F. Lombardi, "A Parallel Decimal Multiplier Using Hybrid Binary Coded Decimal Codes", In IEEE 23-nd Symposium on Computer Arithmetic, pp. 150-155, Aug. 2016.

[13] A.Wahba and H.Fahmy, "Area Efficient and Fast Combined decimal/binary Floating-Point Fused Multiply Add Unit", IEEE, Vol. 66, pp. 226-239, Jun.2016.