IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A PRACTICAL APPROACH OF REDUCTION OF ENERGY/POWER FOR PARALLEL DECIMAL MULTIPLIER

[1]J.RAVISANKAR, [2]B.VENKATARATNAM, [3]B.GOWRI, [4]G.JHANSI

[1]PROFESSOR, [2]M.TECH STUDENT, [3]ASSISTANT PROFESSOR, [4]ASSISTANT PROFESSOR

[1,2,3,4] ECE Department,

[1,2,3,4] KRISHNAVENI ENGINEERING COLLEGE FOR WOMEN, NARASARAOPET, PALNADU (Dt), A.P.INDIA

Abstract: Numerous human-centric applications, including financial services, accounting, tax computation, and currency conversion, rely heavily on decimal computation. A large number of scholars are therefore interested in the development and use of radix-10 arithmetic units. Multiplication isn't only one of the most common but also one of the most complicated and power-hungry operations in fundamental decimal arithmetic. Consequently, this study delves into this matter and examines a generic design technique that minimizes power/energy consumption by localizing switching activity, all while maintaining goal performance. In the same way that Karatsuba's algorithm decomposes a digit multiplier into smaller ones, this approach also allows for varying sizes of the multiplicand and multiplier. Using symmetric and asymmetric divisions of varying sizes, we derive designs with unique properties. The suggested designs surpass prior options for decimal multiplication and provide intriguing area-delay numbers when tested with operands which have 16 digits long, in comparison to traditional binary multipliers. Verilog HDL will be used for its development. To carry out the simulation and synthesis, the Xilinx ISE tool is used.

Index Terms **-** Karatsuba's algorithm, Decimal computation, digital arithmetic, low power design and parallel multiplier.

I. INTRODUCTION

Decimal calculation has made a comeback, even though binary arithmetic functions were implemented quickly and effectively. The advancements in very large scale integration (VLSI) technology, the proliferation of decimal data in human-centric applications (e.g., financial, commercial, scientific, and internet-based), as well as the inability of software implementations to meet high-performance standards all contributed to this revival. Lastly, there is no longer an exact binary code for certain decimal fractions (e.g., 0.2). The first one

enabled the hardware implementation of complicated functions, whereas the other two encouraged designers to include device-implemented decimal arithmetic units to handle the challenge of processing large amounts of data with sufficient accuracy and speed. In light of decimal arithmetic's critical role, the most recent update to the IEEE 754 code for floating point arithmetic includes support for decimal representations and associated operations. Additionally, decimal arithmetic is advancing because to collaborative efforts in both academia and business. A number of processors with specialized decimal arithmetic units have been introduced, one of which being the IBM eServer z900. In contrast, decimal arithmetic methods and hardware units, including decimal addition, have become the subject of a great deal of academic literature.

Hardware that can do decimal floating-point (DFP) calculations is starting to attract attention. Upcoming financial, commercial, while user-oriented applications will have significant performance expectations, and software DFP solutions, although meeting accuracy requirements, are approximately one order of magnitude slower that hardware equivalents. Efforts to establish a norm for decimal arithmetic have also been made. In particular, the DFP arithmetic standards included in the 2008 edition of the IEEE 754 Standard on Floating-Point Arithmetic (IEEE 754-508) may be used in hardware, software, or a hybrid of the two. One example is the IBM z9 architecture, which uses microcode to achieve DFP and uses specialized hardware for the most fundamental operations. The IBM z10 mainframe processor as well as the workstation and server-oriented IBM Power6 microprocessor both come with completely compliant IEEE 754-2008 DFUs. Increasingly efficient methods of decimal multiplication are being demanded by the next generation of high-performance DFUs. The present hardware implementations were underperforming, despite the fact that this remains a crucial and often performed process. For optimal efficiency, the vast majority of binary floating-point units use parallel binary multipliers. The poor performance of expressing decimal values in systems that utilize binary signals combined the difficulty of generating multiplicand multiples make decimal multiplication even harder to accomplish. Because of these problems, producing and reducing incomplete goods is more difficult. So, although decimal adders are practically as efficient as binary adders due to their parallel construction, decimal multipliers for commercial implementations remain sequential. Due to their reliance on iterative algorithms involving multiplication of decimal integers, some implementations exhibit subpar performance. There are a number of recent proposals for methods that either generate partial products or enhance sequential decimal multipliers. New methods for multioperand BCD addition employing tree-like (parallel) structures have been suggested in various other recent research.

Digital Signal Processing (DSP) programs often use complicated mathematical procedures. Improving the fused Add-Multiply (FAM) operator's design to make it more efficient is the main goal of this effort. We look at methods to directly encode the Modified Booth (MB) form of the sum of two integers. Through the use of three distinct methods in FAM designs, they provide a systematic and efficient recoding methodology and investigate them further. The suggested method significantly reduces the FAM unit's power consumption, hardware complexity, and critical latency as compared to FAM devices that employ current recoding algorithms.

II. LITERATURE SURVEY:

High speed speculative multipliers based on speculative carry-save tree by A. Cilardo et al.

The article suggests a new way to construct integer multiplication circuits using speculation, a method that is quicker but sometimes incorrect, and that only uses a multi-cycle error correction circuit when a mistake occurs relatively seldom. Speculative compression, partial product recoding, and partial product partitioning are the 3 stages that make up the new speculative carry-save reduction tree that the suggested speculative multiplier employs. This speculative tree outperforms a traditional tree employing full-adders & half-adders in terms of speed because to the usage of speculative (m:2) counters, where m > 3. Additionally, the study presents a method for automatically selecting appropriate speculative counters by considering both the mistake probability and latency. An error correcting circuit including a fast speculative carry-propagate adder finish off the speculative tree. Speculative multipliers for different operand lengths have been generated with the UMC 65 nm library. Whenever a lot of speed is needed, speculation works better than traditional multipliers. In situations when a high speed operation is necessary, speculative multipliers not only enable obtaining a greater speed than traditional equivalents, but they are also very effective in terms of power dissipation.

Multipliers of two's complement have applications in many contexts. Here we provide a method that, without increasing the latency of the partial product generation step, may decrease the maximum height for the partial product array formed by a radix-4 Modified Booth Encoded multiplier by one row. Because of this cut, normal layouts and the partial product array might be able to be compressed more quickly. While this method is useful in multiplier design generally, it has the utmost importance in high-performance embedded cores' low bit-width two's complement multipliers. Any size square or m \times n rectangle multipliers may be handled by extending the suggested approach, which is also applicable to larger radix encodings. Results from a preliminary theoretical analysis plus logic synthesis demonstrated the suggested approach's effectiveness in area and latency, therefore we compared it to other potential alternatives.

Design of fixed-width multipliers with linear compensation function by N. Petra et al

By delving deeply into the impact of quantization of coefficients, this research zeroes down on fixed-width multipliers using linear compensation functions. By manipulating the quantization technique, one may derive new fixed-width multiplier topologies that exhibit distinct trade-offs between hardware complexity and accuracy. The two most successful topologies are singled out. While the initial one relies on uniform quantization of coefficients, the second one employs a nonuniform quantization approach. Compared to earlier solutions, the new fixed-width multiplier topologies are more accurate and get closer to the theoretical lower limit.

Sign-magnitude encoding for efficient VLSI realization of decimal multiplication by S. Gorgin and G. Jaberipur

It is usual practice to choose intermediate partial products (IPPs) among a list of precomputed radix-10 X multiples when performing the complicated operation of decimal X × Y multiplication. By converting the digits of Y into the one-hot representation of the signed digits in the interval [-5,5], some tasks only need [0,

5] × X. There will be an additional IPP, but the selection rationale will be lessened. IPPs are often represented using Two's complement signed-digit (TCSD) encoding, which requires dynamic negation to generate the recoded digits of Y in [-5, -1] via one xor per bit of X multiples. The research presented here shows that for 16-digit operands, circuitry is possible to begin a partial product reduction (PPR) using 16 IPPs which improve the VLSI regularity, even if 17 IPPs are generated. In addition, by encoding precomputed multiples using sign-magnitude signed-digit (SMSD) notation, we are able to save 75% of negating xors. We provide an efficient adder to calculate the first-level PPR that takes two SMSD numbers as input and represents their total using TCSD encoding. After that, the final binary-coded decimal product is obtained by combining the results of two TCSD accumulated partial products that are the result of a multilayer TCSD 2:1 reduction. These products are then subjected to a unique early begun conversion method. For this reason, we synthesize a very large scale integration (VLSI) version of a 16 × 16-digit parallel decimal multiplier, and our assessments reveal a slight performance boost compared to earlier equivalent implementations.

A high-frequency decimal Multiplier by R. D. Kenney, M. J. Schulte, and M. A. Erle

The increasing significance of commercial, financial, other Internet-based applications that handle decimal data has led to decimal arithmetic regaining favor among the computer world. An iterative decimal multiplier that scales well for large operand sizes and functions at high clock frequencies is shown in this study. This two-stage iterative multiplier architecture is very quick because the multiplier employs a new decimal representation with intermediate results. The clock frequencies used by decimal multipliers, typically are created from a library of 0.11 micron CMOS regular cells, are close to 2 GHz. A new multiplication may start every (n+1) cycles depending on the suggested architecture, which has a latency of (n+8) cycles when multiplying two n-digit BCD operands.

III. KARATSUBA'S ALGORITHM

When used to the Karatsuba algorithm, this "Divide and conquer" strategy yields an impressive asymptotic speedup above a long-lost approach. To multiply two numbers with n digits, the conventional classroom approach calls for Θ(n2) digit operations. We will demonstrate that the issue can be solved in O(nlog 3) digit operations using a straightforward recursive approach. (Note: log 3 ≈ 1.58.). When looking towards the asymptotic growth rate number the total amount of digit-operations that indicates a significant reduction. In pseudo code, we detail the process. To avoid worrying about rounding, we shall presume that the total number for digits corresponds to a power of 2. The use of starting zeros to pad the amount supports this assumption; doing so will raise the value of n much just over a factor of 2, which is insignificant for our predictions.

Procedure Karatsuba(X, Y)

Input: X, Y : n-digit integers. Output: the product P := XY . Comment: We assume n is a power of 2.

1. if n = 1 then use multiplication table to find P := XY
2. else split X, Y in half:
3. X =: $10^{n/2}X_1 + X_2$
4. Y =: $10^{n/2}Y_1 + Y_2$
5. Comment: X_1, X_2, Y_1, Y_2 each have n/2 digits

6. U := Karatsuba(X_1, Y_1)

7. V := Karatsuba(X_2, Y_2)

8. W := Karatsuba($X_1 - X_2$, $Y_1 - Y_2$)

9. Z := U + V − W

10. P := $10^n U + 10^{n/2} Z + V$

11. Comment: So U = X_1Y_1, V = X_2Y_2, W = $(X_1 - X_2)(Y_1 - Y_2)$, and therefore Z = $X_1Y_2 + X_2Y_1$. Finally we conclude that P = $10^n X_1Y_1 + 10^{n/2}(X_1Y_2 + X_2Y_1) + X_2Y_2 = XY$.

12. return P

Analysis. In order to complete its work, this algorithm repeatedly calls itself, resulting in a recursive algorithm. The total number of digit-multiplications needed using the Karatsuba method to produce two n-digit integers (n = 2k) is denoted by M (n) (line 1).

Since the process itself is called three times on numbers with n/2 digits in lines 6, 7, and 8, the result is M (n) = 3M (n/2). (1)

The following is a straightforward solution to this problem as it is a simple recurrence. Since M (n/2) = 3M (n/4), we may deduce that M (n) = 9M (n/4), by plugging M (n/2) into equation (1). It can be deduced from induction on i that for any i (i < k), M (n) = 3iM (n/2i), and continuing along the same way, we get that M (n) = 27M (n/8).

With i= k, we get 3kM(n/2k), which is equal to 3kM(1), which is equal to 3k. Keep in mind that k = log n (remember that log here means base-2 logarithm), consequently M (n) = 2log So, M(n) equals 2k. The value of log 3 is equal to 2-k.nlog 3 is equal to log 3. It seems that either increased the number of additions (lines 9, 10), decreased the number for digit-multiplications (to nlog 3), or anything along those lines. Despite first impressions, the approach saves just as much work when it comes to the overall amount of digit-operations (additions and multiplications combined). Let T (n) be the entire number of digit-operations needed by the Karatsuba algorithm, which includes additions, multiplications, and bookkeeping operations such as copying digits and maintaining connections. This will allow you to observe this. After that

T (n) = 3T (n/2) + O(n) (2)

Like in the previous case, the term 3T (n/2) is derived from lines 6, 7, and 8. The extra O(n) term represents the number many digit-additions needed to execute the operations in lines 9 and 10. Includes accounting expenses in the O(n) period as well.

The analysis of recurrences with the type (2) will be covered later on in the course. The rate of increase is unaffected by the additive O(n) term, and the outcome will remain the same.

T (n) = O(nlog 3). (3)

III. THE PROPOSED METHOD

One famous divide-and-conquer method for speeding up multiplication of big numbers utilizes the Karatsuba algorithm. Many suggested high-speed multipliers are based on this method. This work, on the other hand, is concerned with partitioning-based power reduction for decimal multiplication, such as the Karatsuba method. It offers sufficient granularity when localizing switching activity and employs smaller multipliers, often

known as multiplier cells. However, it is important to thoroughly investigate some matters, such as the fundamental multiplication method and the amount of space of the various multiplier cells that are used.

DECIMAL MULTIPLICATION VIA PARTITIONING

To speed up the multiplication of big numbers, the Karatsuba algorithm uses divide and conquer, as shown above. Its first use came in binary multiplication. In order to calculate P = X x Y using this decimal multiplication procedure, we may recursively split the operands into XH and XL, and YH and YL, correspondingly. Regarding this, those that are most important components are shown by XH and YH, while the least significant parts are shown by XL and YL. The two operands may be represented in below Equation (2) simply presuming k = 2n and equal partitioning.

$$X = 10^n \sum_{i=0}^{n-1} x_{i+n} 10^i + \sum_{i=0}^{n-1} x_i 10^i = X_H 10^n + X_L$$
$$Y = 10^n \sum_{i=0}^{n-1} y_{i+n} 10^i + \sum_{i=0}^{n-1} y_i 10^i = Y_H 10^n + Y_L$$

---------- Equation (2)

The below Equation is employed to generate the P D X x Y product following partitioning, using the Karatsuba multiplication technique.

$$P = \left(X_H 10^n + X_L\right)\left(Y_H 10^n + Y_L\right)$$
$$= 10^{2n} X_H Y_H + 10^n \left(X_H Y_L + X_L Y_H\right) + Y_L Y_L$$

---------- Equation (3)

Four multiplications of decimals with nXn digits and two additions with 2n and 3n digits are required to implement the above Equation (3). It will be possible to iteratively run this method until it reaches the 1X1 digit multiplication.

3.1 SYMMETRIC APPROACH

We take into account three distinct operand partitioning sizes—two, four, and eight—for symmetric architectures. Equation (3) states that a 16 X 16 digit multiplier may be constructed from four 8 X 8 digit multipliers using the initial layout, which involves dividing the multiplier each multiplicand equal two equal portions.

Before moving through the latter stages of reduction and conversion, it is necessary to align the results of the multiplier cells. Figure 1 shows this alignment, which results in the smallest significant component and the final result being the eight bits from the least significant part, requiring any further calculation. Nevertheless, a multi-operand addition is required for positions 8–23. Each of the eight bits and the most important element also need an increment operation (increment with carry out of multi-operand addition). It is important to note how the multiplier cell method is crucial since it affects the primary multiplier's low-

level design and detail design. So, we've covered the algorithm's multiplier cells and their associated important features.

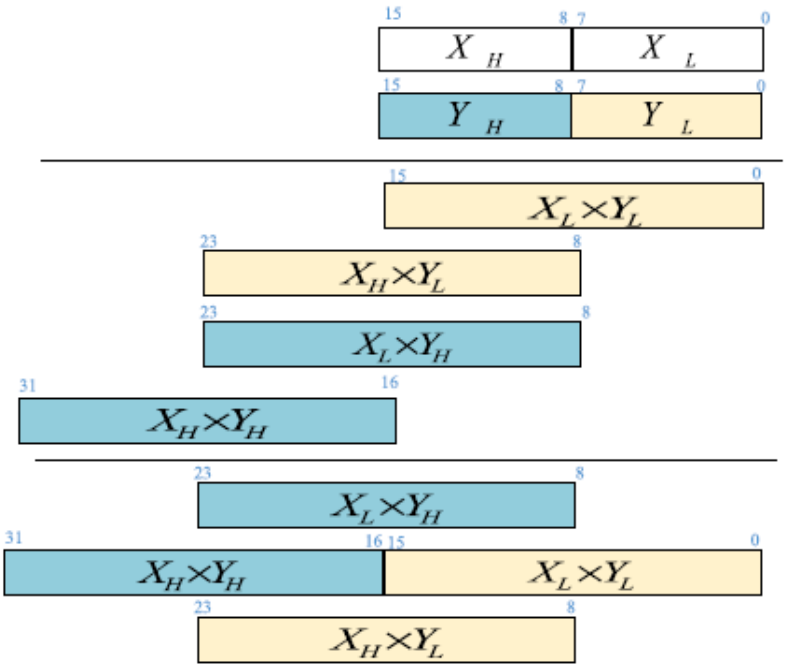

FIGURE 3.1. The arrangement of the 16 X 16-digit multiplier by four 8 X 8 multipliers (mult.16-8).

Recursively replacing every multiplier cell with an even smaller one is possible using computing divide-and-conquer principle. As earlier stated, four 2n X 2n-digit multipliers are required for a 4n X 4n-digit multiplier. Additional partitioning allows for the implementation of four n X n-digit multipliers for every 2n X 2n-digit multiplier. As seen in Figure 2, these considerations lead us to believe that 16 4 X 4-digit multipliers may be used to build a 16 X 16-digit multiplier.

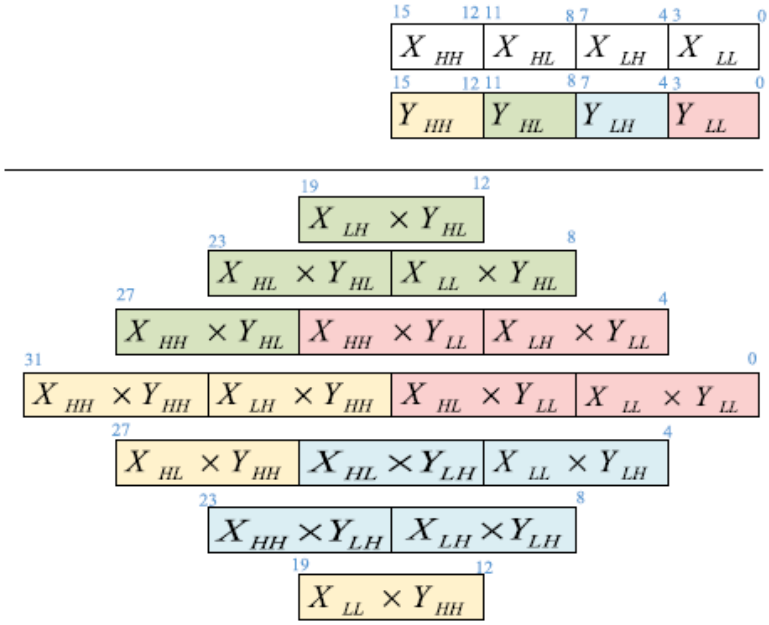

FIGURE 3.2. The arrangement of the 16 X 16-digit multiplier by sixteen 4 X 4-digit multipliers (mult16-4).

IV. Its partial outcomes provide digit-by-digit multipliers near the extreme of partitioning, that in turn offers a parallel multiplier. That 2X2-digit multiplier is defined as the most tiny multiplier cell while all the state-of-the-art multipliers employ pre-computed multiplies depending on background data. We may build a 16X16-digit multiplier out of sixty-four 2_2-digit multipliers, as seen in Figure 3, by dividing the operands into eight equal pieces and applying this idea.

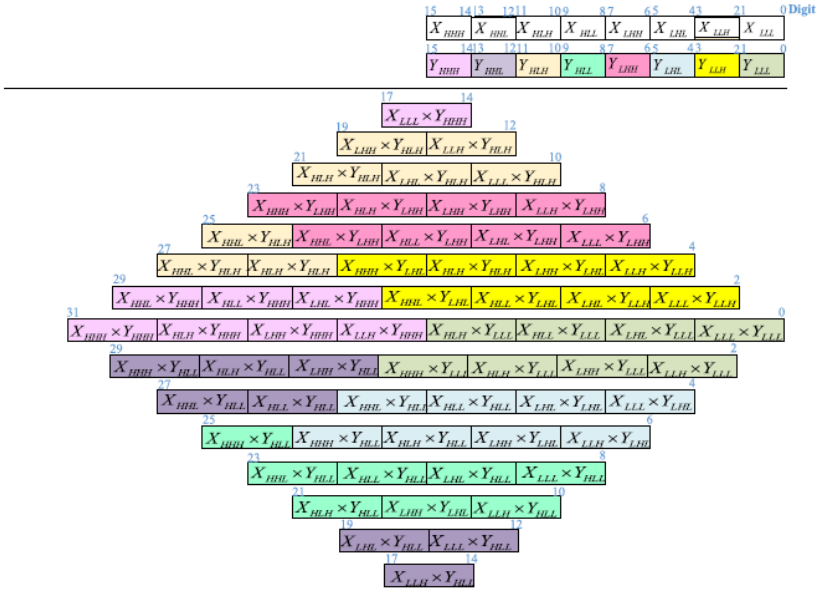

FIGURE 3.3. The arrangement of the 16 X 16-digit multiplier by sixty-four 2 _ 2-digit multipliers.

Two major considerations should be made while examining the aforementioned designs. The depth within the multi-operand adder and the format that the multiplier cells' output are the two variables in question. A multi-operand adder with depths of 3, 7, and 15 is shown in Figures 3, 4, and 5, correspondingly. Clearly, finer granularity is achieved by employing smaller multiplier cells. Having said that, it does raise the usage of area. As a result, reducing the size overall multiplier cells and their area usage requires a thorough examination that includes monitoring of power dissipation. Regarding the second point, as shown in Figure 2, the BCD format is utilized for the final result for multiplier cells. This format is supplied when a conversion from redundant to non-redundant data. Because a multioperand adder takes its input from multiplier cells, this wasteful carry propagation is superfluous. This adder must, of course, take in duplicate data. This problem is associated with the multiplier cell method.

3.2 ASYMMETRIC APPROACH

V. Multiplier cells may be of varying sizes, as previously stated. Because of imbalanced delay routes, the overall delay associated with the main multiplier can be increased by a larger multiplier cell, even if it has greater delay. Section 5 provides the experimental findings, however it's worth noting that an asymmetric technique may demonstrate significant advantage for a given input pattern having unique statistical features. Given the vast number of alternative partitioning schemes for asymmetric designs, our suggested technique

highlights the benefits of this sort of design by analyzing a simple partitioning to demonstrate a particular pattern of energy usage.

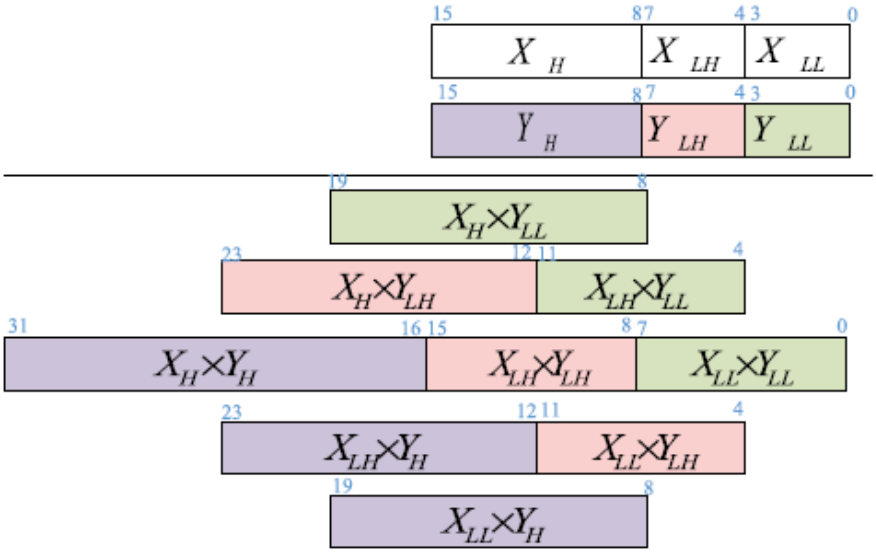

FIGURE 3.4. The arrangement of the 16 X 16-digit multiplier with asymmetric partitioning (mult16-8-4).

Fig. 4 shows the asymmetrical partitioning within the 16_16-digit multiplier. Four 4 X 4-digit multipliers, two 4 X 8-digit multipliers, two 8 X 4-digit multipliers, along with a 8 X 8-digit multiplier make up this multiplier.

IV SIMULATION RESULTS & SYNTHESIS REPORT

4.1 Simulation Result of Multiplier:

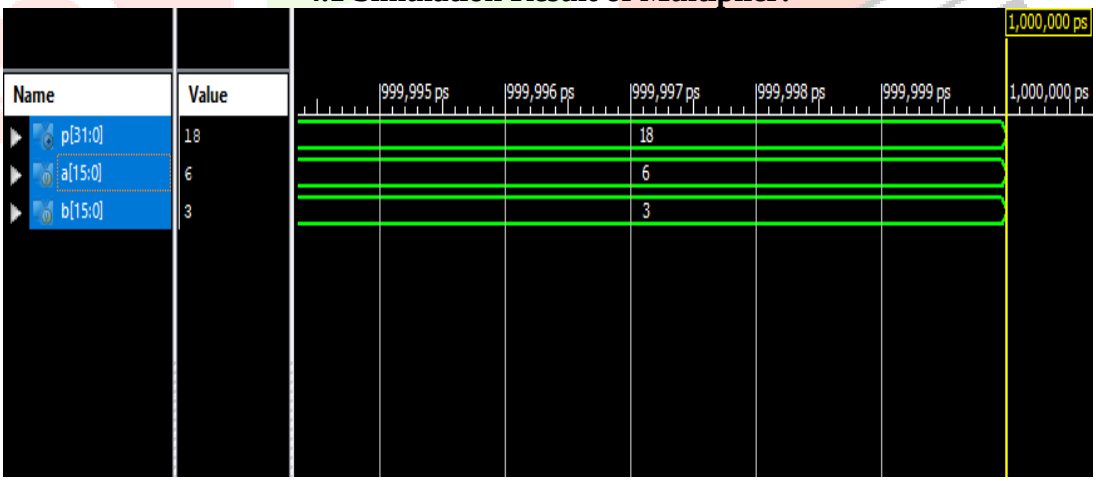

Fig 4.1 Simulation Result of Multiplier

Here we can give the inputs as A=6, B=3, then the final output is 18.

4.2 Block Diagram of Multiplier:

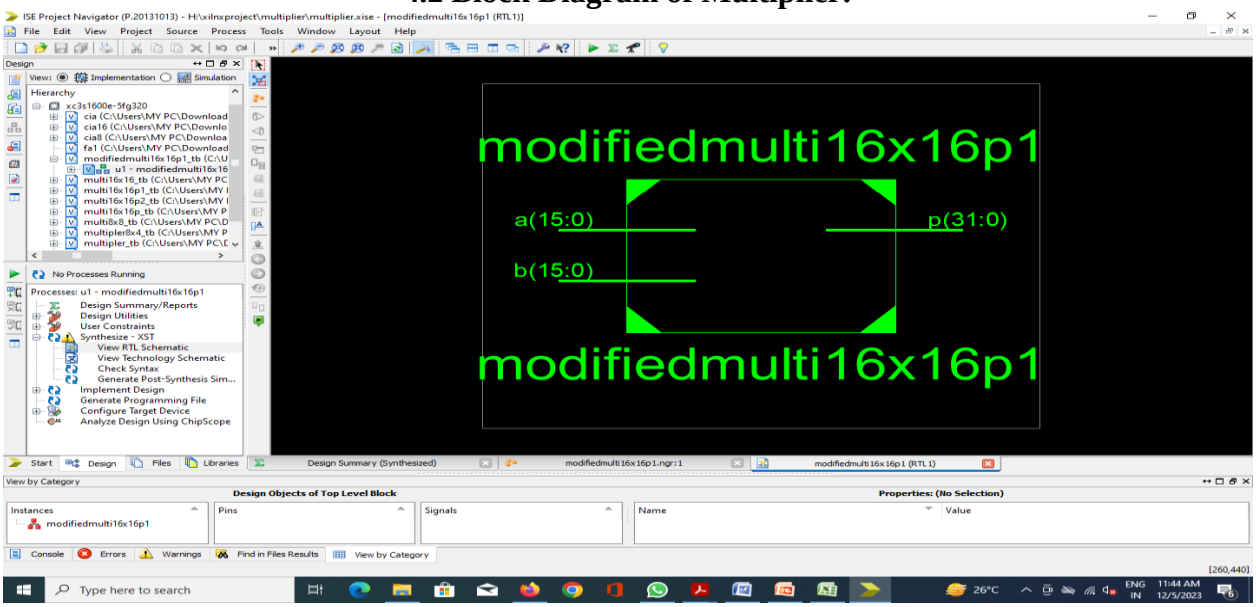

Fig 4.2 Block diagram

4.3 RTL SCHEMATIC:

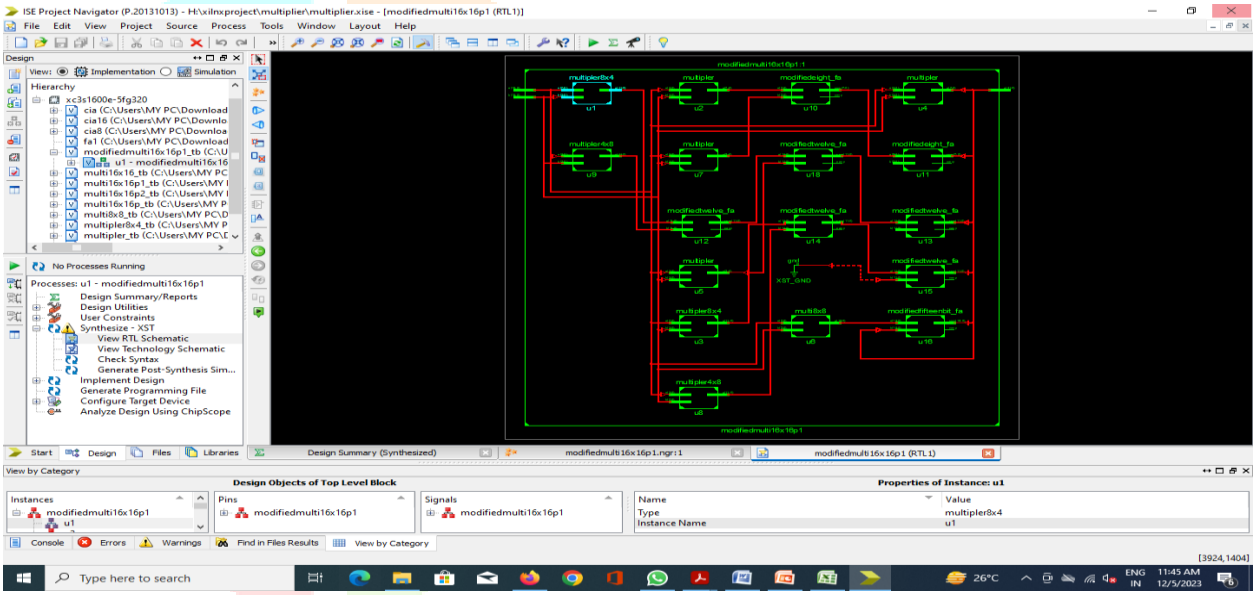

Fig 4.3 Internal diagram of RTL schematic

4.4 Estimation of power:

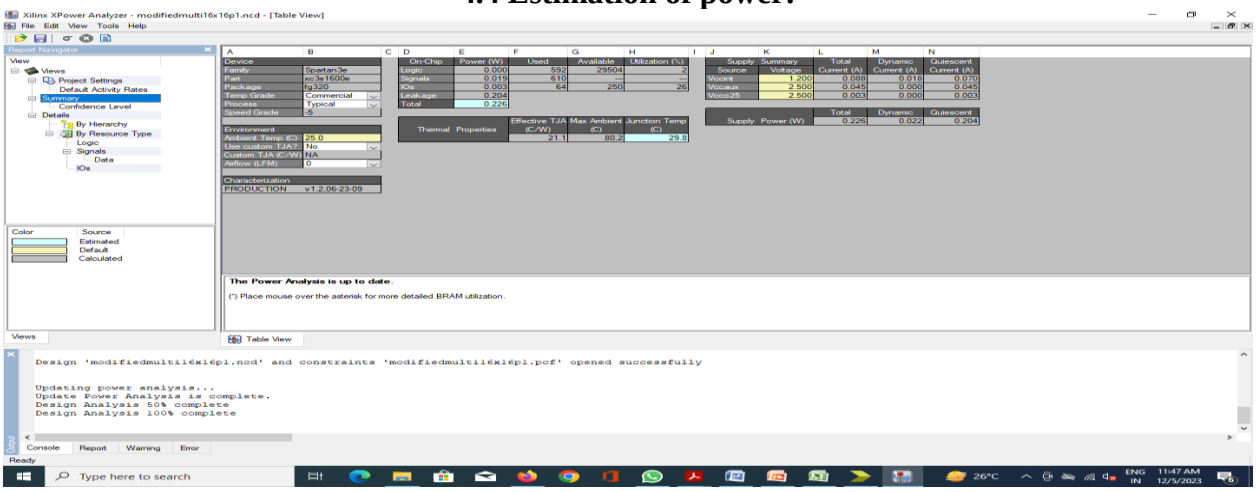

Fig 4.4 Estimation of power

Fig 4.4 Estimates power using approximate multiplier and it is 0.226W power.

4.5 Estimation of Delay:

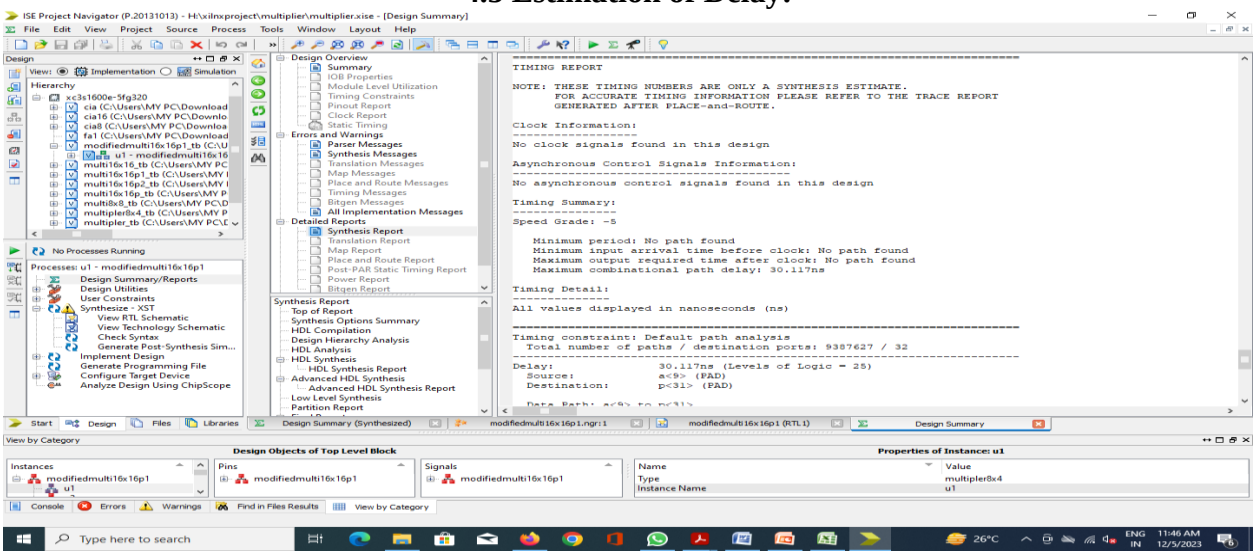

Fig 4.5 Estimation of Delay

Fig 4.5 Estimates delay of approximate multiplier and it is 30.11ns delay.

4.6 Estimation of Area:

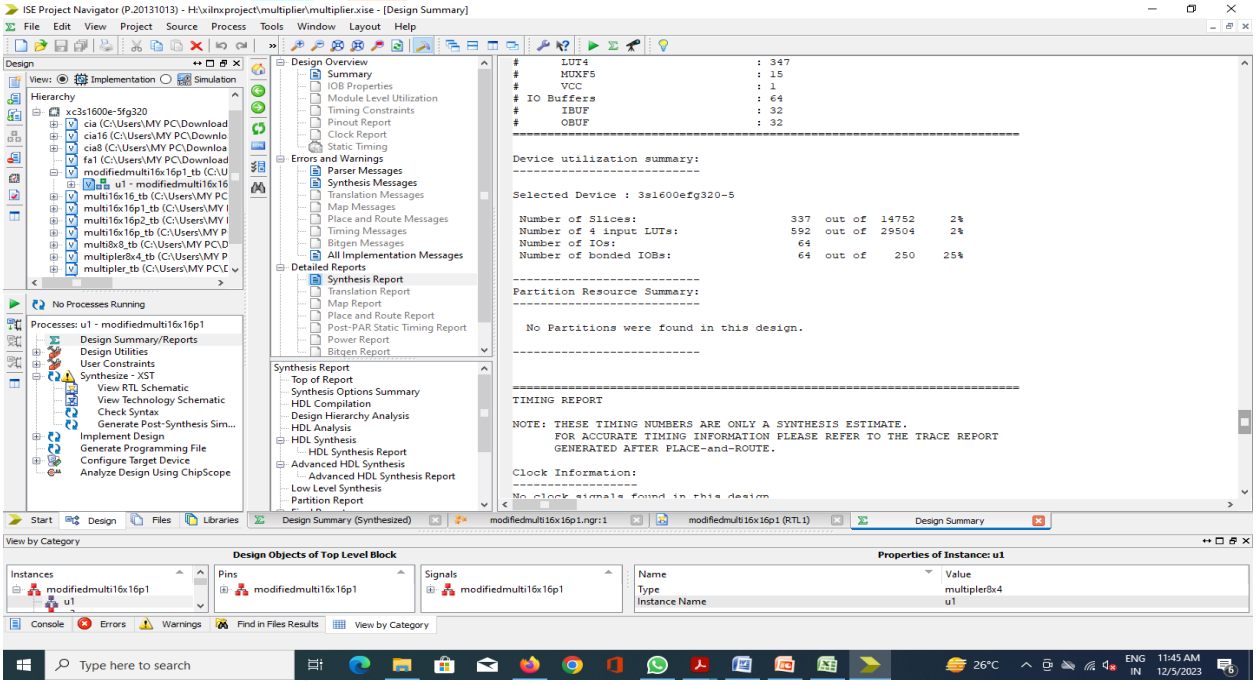

Fig 4.6 Estimation of Area

4. 7 COMPRESSION TABLE

	extension	8*4 and 4*8	8*8	4bit	2bit
delay	30.117	33.170	49.938	42.224	36.170
area	592	520	639	644	541

V.CONCLUSION & FUTURE SCOPE

In this paper, we investigate a design technique for decreasing the power used by decimal multipliers, since decimal calculation is important. The proposed approach utilizes partitioning, which combines the smaller ones to form a larger multiplier. Xilinx ise was used for synthesis, and structural verilog was utilized to implement the suggested designs and the first multiplier, resulting in multiplier cells. Experiments show that when every one of the input signals were active, delays are reduced.

REFERENCES

[1] S. Kuang, J. Wang, and C. Guo, “Modified booth multipliers with a regular partial product array,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 56, no. 5, pp. 404–408, May 2009.

[2] F. Lamberti et al., “Reducing the computation time in (short bit-width) twos complement multipliers,” IEEE Trans. Comput., vol. 60, no. 2, pp. 148–156, Feb. 2011.

[3] N. Petra et al., “Design of fixed-width multipliers with linear compensation function,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 58, no. 5, pp. 947– 960, May 2011.

[4] S. Galal et al., “FPU generator for design space exploration,” in Proc. 21st IEEE Symp. Comput. Arithmetic (ARITH), Apr. 2013, pp. 25–34.

[5] K. Tsoumanis et al., “An optimized modified booth recoder for efficient design of the add-multiply operator,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 61, no. 4, pp. 1133–1143, Apr. 2014.

[6] M. F. Cowlishaw, ``Decimal _oating-point: Algorism for computers," in Proc. 16th IEEE Symp. Comput. Arithmetic, Jun. 2003, pp. 104_111.

[7] IEEE Standards Committee, Standard 754-2008, 2008, doi: 10.1109/IEEESTD.2008.4610935.

[8] Draft IEEE Standard for Floating-Point Arithmetic, Standard P754/D2.50, Apr. 2019.

[9] F. Y. Busaba, C. A. Krygowski, W. H. Li, E. M. Schwarz, and S. R. Carlough, ``The IBM z900 decimal arithmetic unit," in Proc. 25th Asilomar Conf. Signals, Syst. Comput., 2001, pp. 1335_1339.

[10] L. Eisen, J. Ward, H. Tast, and N. Mading, ``IBM POWER6 accelerators: VMX and DFU," IBM J. Res. Develop., vol. 51, no. 6, pp. 633_684, 2007.

[11] C. F.Webb, ``IBM Z10: The next-generation mainframe microprocessor," IEEE Micro, vol. 28, no. 2, pp. 19_29, Mar. 2008.

[12] A. Vazquez and E. Antelo, ``Conditional speculative decimal addition," in Proc. 7th Conf. Real Numbers Comput., Jul. 2006, pp. 47_57.

[13] R. D. Kenney and M. J. Schulte, ``High-speed multi operand decimal adders," IEEE Trans. Comput., vol. 54, no. 8, pp. 953_963, 2005.

[14] M. A. Erle and M. J. Schulte, ``Decimal multiplication via carry-save addition," in Proc. 14th IEEE Int. Conf. Appl.-Speci_c Syst., Archit., Processors, 2003, pp. 348_358.

[15] S. Gorgin and G. Jaberipur, ``Sign-magnitude encoding for efficient VLSI realization of decimal multiplication," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 25, no. 1, pp. 75_86, Jan. 2017, doi: 10.1109/TVLSI.2016.2579667.

[16] A. Kaivani, A. Hosseiny, and G. Jaberipur, ``Improving the speed of decimal division," IET Comput. Digit. Techn., vol. 5, no. 5, pp. 393_404, 2011.

[17] A. Hosseiny and G. Jaberipur, ``Decimal Goldschmidt: A hardware algorithm for radix-10 division," Comput. Electr. Eng., vol. 53, pp. 40_55, Jul. 2016.