ENGINEERING IN ADVANCED RESEARCH SCIENCE AND TECHNOLOGY

UGC Care Group-I
Vol.03, Issue.02
December-2022
Pages: -1165-1177

LATENCY OPTIMIZED FINITE-FIELD MULTIPLICATION USING LOW DENSE MODIFIED URDHVA TIRYAGBHYAM

[1]Govad keerthana, [2]R.S.V.S Aravind, [3]Dr.B.Raja Rao
[1]M. tech, Dept. of ECE, Eluru College of Engineering and Technology, ELURU, AP
[2]Assistant Professor, Dept. of ECE, Eluru College of Engineering and Technology, ELURU, AP
[3]Professor & H.O.D, Dept. of ECE, Eluru College of Engineering and Technology, ELURU, AP

ABSTRACT: Arithmetic in Finite/Galois field is a major aspect for many applications such as error correcting code and cryptography. Addition and multiplication are the two basic operations in the finite field GF. The complexity (space) analysis and efficient FPGA implementation of bit parallel Karatsuba Multiplier over GF (2m) is presented. This is especially interesting for high performance systems because of its carry free property. Using Karatsuba multiplier we can improve the performance of the process. This paper proposes an optimised polynomial multiplication for compact digital architectures. This concept proposes novel optimisations for the most computationally intensive part of lattice-based cryptography constructions, i.e., the polynomial multiplier, targeting the high speed hardware platform. In ECC systems, polynomial multiplication is considered to be the most slow and area consuming operation. This article proposes a novel hardware architecture for efficient field-programmable gate array (FPGA) implementation of Finitefield multipliers for ECC. The OKA is a speed-optimized version of the original Karatsuba. In this method, to improve the longest path delay, inputs are split into odd and even orders instead of the high and low parts. Proposed architecture shows an example DFG for FPGA implementation of a 2-bit and a 4-bit binary polynomial multiplier using six-input LUTs. As an enhancement of this work, the performance of Karatsuba algorithm can be compared with the performance of the other algorithms developed for performing multiplication operation. In addition, an algorithm can be developed for increasing the efficiency of multiplication operation and reducing the cost , area and latency of process.

INTRODUCTION: Published in 1962 [2], Karatsuba-Ofman's algorithm (KOA) was the first integer multiplication method that broke the quadratic complexity barrier in positional number systems. Due to its simplicity, its polynomial version is widely adopted to design VLSI parallel multipliers in GF(2n)-based cryptosystems [13]-[34]. Two parameters are often used to measure the performance of a GF(2n) parallel multiplier, namely, the space and time complexities. The space complexity is represented in terms of the total number of 2-input XOR and AND gates used. The corresponding time complexity is given in terms of the maximum delay faced by a signal due to these XOR and AND gates. Symbols "TA" and "TX" are often used to represent the delays of one 2-input AND gate and one 2-input XOR gate, respectively. The existing bit parallel GF(2n) multipliers may be simply classified into the following three categories according to the asymptotic space complexity of the multiplication algorithm: quadratic, sub quadratic and hybrid multipliers. A number of quadratic multipliers have

been proposed in the literature in which different basis representations of GF(2n) elements are used, e.g., polynomial, shifted polynomial, normal, dual, weakly dual, and triangular bases. Their time complexities are lower than those of sub quadratic multipliers. The main advantage of sub quadratic multipliers is that their low asymptotic space complexities make it possible to implement VLSI multipliers for large values of n. But when the size of operands is small, e.g., 32-bit, the space complexity may not remain as the critical factor considered by a cryptographic processor designer. Instead, the computational speed becomes the key factor. These multipliers first perform a few KOA iterations to reduce the whole space complexities, and then a quadratic multiplication algorithm on small input operands to achieve relatively high speed performance. By selecting different stop conditions for the KOA iterations, the hybrid approach can provide a trade-off between the time and space complexities. Its space complexity is lower than that of the original KOA multiplier.

LITERATURE SURVEY:C. P. Rentería-Mejía et.al [1] proposed a Hardware Design of FFT Polynomial Multipliers. In this paper, they present the design of two FFT polynomial multipliers using parallel and sequential architectures. Parallel and sequential polynomial multipliers were optimized for throughput and area resources, respectively. The designs are described in generic structural VHDL, synthesized on the Stratix EP4SGX230KF40C2 using Quartus II V. 13, and verified using SignalTap. The hardware synthesis and performance results show that the designed multipliers present a good area throughput trade-off and they are suitable for high-performance scientific computing applications. Their work presents the design of two polynomial multipliers based on FFT. In this case, they used FFT based on complex fixed-point computations and R22SDF architecture. Parallel and sequential polynomial multipliers were optimized for throughput and area resources, respectively. Also, the designed multipliers were parameterized for polynomials of 8, 16, 32, 64, 128, 256 and 512 coefficients. The synthesis results show that the designed polynomial multipliers use few area resources and have a good throughput. The parallel polynomial multiplier uses 53 % more resources and its throughput is in average 1.81 times bigger than the sequential polynomial multiplier. Also, the designed multipliers carry out the polynomial multiplication in less time than the corresponding software simulation in Maple 15, which was performed on an Intel Core i7-3770 CPU @ 3.40 G taking into account the synthesis and hardware verification results, they conclude that the designed multipliers are suitable for high-performance scientific computing applications [1].

Table 1. Performance results for pp and sp multipliers [1]

N	No of cycles		Mult. Time (µs)		Throughput (Gb/s)	
	PP	SP	PP	SP	PP	SP
8	40	68	0.64	1.12	1.05	0.60
16	74	127	1.21	2.11	1.19	0.68
32	140	242	2.30	4.24	1.29	0.70
64	270	469	4.43	7.76	1.37	0.78
128	528	920	8.74	15.86	1.39	0.77
256	1042	1819	17.29	31.82	1.42	0.77
512	2068	3614	34.31	63.44	1.43	0.77
1024	4118	7201	68.19	126.22	1.44	0.78

Lo Sing Cheng et.al [2] presents an Efficient FPGA Implementation of FFT Based Multipliers. Finite field multiplication is one of the most useful arithmetic operations and has applications in many areas such as signal

processing, coding theory and cryptography. However, it is also one of the most time consuming operation in both software and hardware, which makes it pertinent to develop a fast and efficient implementation.

EXISTING METHOD:

OVERLAP-FREE KARATSUBA ALGORITHM

The OKA is a speed-optimized version of the original Karatsuba. In this method, to improve the longest path delay, inputs are split into odd and even orders instead of the high and low parts. Once more, it is assumed that *A(x)* and *B(x)* are two polynomial in *GF(2n)* and $n = 2m$. These polynomials could be rewritten as

$$A(X) = \sum_{i=0}^{m-1} a_{2i}\ x^{2i} + \sum_{i=0}^{m-1} a_{2i+1}\ x^{2i+1}$$
$$B(x) = \sum_{i=0}^{m-1} b_{2i}\ x^{2i} + \sum_{i=0}^{m-1} b_{2i+1}\ x^{2i+1}.$$

Assuming that $y = x^2$, above equations can be rewritten as

$$A(x) = \sum_{i=0}^{m-1} a_{2i}\ y^i + x \sum_{i=0}^{m-1} a_{2i+1}\ y^i = A_e(y) + xA_o(y)$$
$$B(x) = \sum_{i=0}^{m-1} b_{2i}\ y^i + x \sum_{i=0}^{m-1} b_{2i+1}\ y^i = B_e(y) + xB_o(y)$$

where *Ae* and *Be* include the even order and *Ao* and *Bo* include the odd-order terms of polynomials *A(X)* and *B(x)*, respectively. Following a similar approach to the KA, the polynomial multiplication could be calculated as:

$$\begin{aligned} A(x)B(x) &= (A_e(y) + xA_o(y)) \times (B_e(y) + xB_o(y)) \\ &= G_2(y)y + [G_1(y) - G_2(y) - G_0(y)]x + G_0(y) \end{aligned}$$

where

$$G_0 = A_eB_e$$
$$G_1 = (A_o + A_e)(B_o + B_e)$$
$$G_2 = A_oB_o.$$

Similar to the KA, the overlap-free algorithm also uses three submultipliers. However, term $G0(y) + yG2(y)$ has only odd exponents (x^{2n+1}), and term $G1(y)$ contains only terms with even exponents (x^{2n}). Therefore, there is no overlap between the components of these two terms, which allows removing an XOR gate from the critical path of the Karatsuba multiplier. As an example, the DFG of a 4-bit OKA multiplier is shown in Fig. 3. It should be noted that the DFG for a first-order OKA multiplier is similar to that of the KA, as shown in Fig. (a). As shown in Fig, for the 4-bit multiplier, the critical path of overlap-free is one XOR gate delay shorter than a Karatsuba multiplier. By solving the recursive equation for area and space requirements, the estimated values of the OKA implementation are as follows:

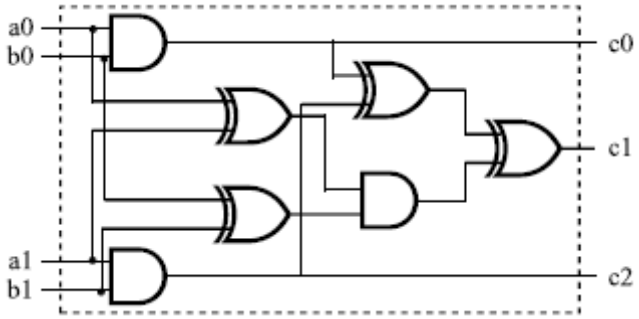

Fig1. DFG for hardware implementation of (a) 2-bit

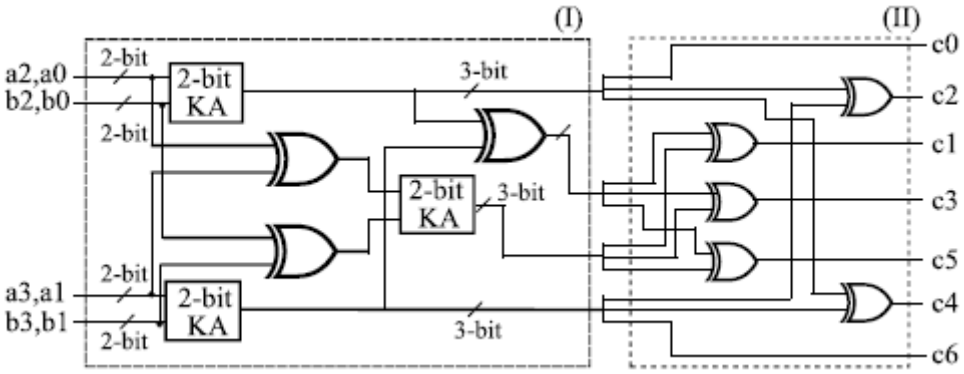

Fig: 2 Bit Karatsuba multiplier

LUT BASED IMPLEMENTATION:

The total number of gates for hardware implementation of the binary polynomial multiplication algorithms for different operand sizes is presented in Fig. 4(a). From this figure, it can be observed that considering small operand sizes, the number of gates required for implementing CAs is lower than that of the KA. However, as the operand size grows, the number of gates for implementing CA becomes substantially higher than Karatsuba and overlap-free. As an example, for operand size of 409 bits, the CA requires almost 163% more gates than the Karatsuba or overlap-free.

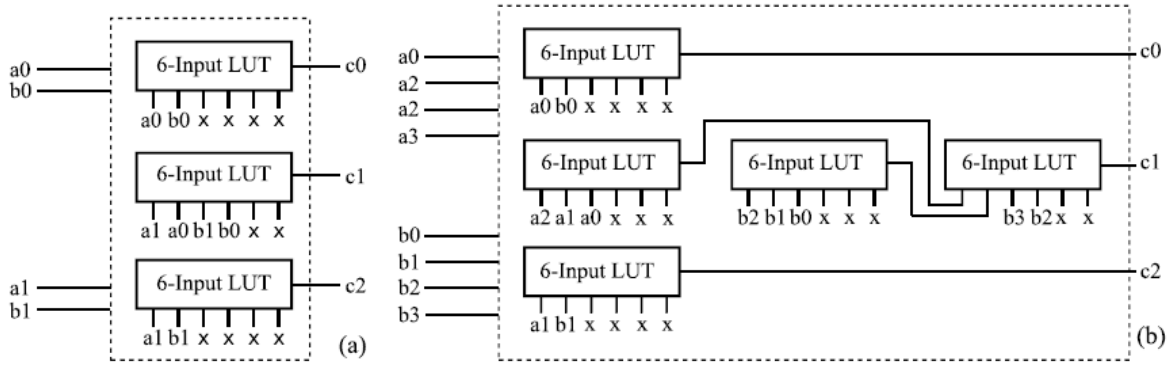

Fig. 3. DFG for FPGA implementation of (a) 2- and (b) 4-bit binary polynomial multipliers using six-input LUTs.

Regardless of the different DFGs, FPGA implementation of all methods for 2- and 8-bit multipliers is the same. It is worth noting that this is a simple demonstration and actual architecture and routing is more complex.

Another interesting point is that the number of stages required to implement recursive multipliers, including KA and OKA, increases logarithmically with operand size rather than linearly. As an instance, in the case of the 233 bit, the first four stages recursively perform multiplication down to 15-bit multipliers. However, performing a 15-bit multiplication requires another four stages. It is also worth noting that the overall delay of these multipliers is determined by their corresponding number of stages. To compare the efficiency of algorithms, area–delay product (ADP) was calculated for all algorithms and plotted in Fig. 4(c) where, on average, overlap-free has the minimum and conventional has the highest ADP. Since our target platform is FPGAs, not digital gate-based devices, we investigated on-FPGA time and space analysis of these algorithms and validated the outcome by implementing algorithms. When it comes to the FPGA devices, the building blocks constructing most functions are lookup tables (LUTs) and not combinational gates. The LUTs are considered as universal gates where any function could be represented. Therefore, in order to accurately estimate the complexity and delay analysis, these structures should be implemented on the FPGA using the LUTs. Fig. 6 shows an example DFG for FPGA implementation of a 2-bit and a 4-bit binary polynomial multiplier using six-input LUTs. As shown in this figure, irrespective of the number of gates and the difference in the DFGs, the LUT-based implementations are similar. The difference between LUT implementation of these algorithms in terms of performance and the number of LUTs becomes distinct as the operand size increases. Therefore, the theoretical estimations that are conventionally used to evaluate the efficiency of the algorithms cannot be simply extended to their actual FPGA implementation.

There are techniques to estimate the number of LUTs and delay for FPGA implementation of combinational circuits. However, in a pragmatic approach, all algorithms mentioned above for various operand sizes were implemented on FPGA.

PROPOSED METHOD:

VEDIC MULTIPLICATION:

VEDIC MULTIPLIER FOR 2X2 BIT MODULE:

The method is explained below for two, 2 bit numbers A and B where $A = a1a0$ and $B = b1b0$ as shown in Fig. 2. Firstly, the least significant bits are multiplied which gives the least significant bit of the final product (vertical). Then, the LSB of the multiplicand is multiplied with the next higher bit of the multiplier and added with, the product of LSB of multiplier and next higher bit of the multiplicand (crosswise). The sum gives second bit of the final product and the carry is added with the partial product obtained by multiplying the most significant bits to give the sum and carry. The sum is the third corresponding bit and carry becomes the fourth bit Of the finel product. The 2X2 Vedic multiplier module is implemented using four input AND gates & two half-adders which is displayed in its block diagram in Fig. 3. It is found that the hardware architecture of 2x2 bit Vedic multiplier is same as the hardware architecture of 2x2 bit conventional Array Multiplier [2]. Hence it is concluded that multiplication of 2 bit binary numbers by Vedic method does not made significant effect in improvement of the multiplier‟s efficiency. Very precisely we can state that the total delay is only 2-half adder delays, after final bit products are generated, which is very similar to Array multiplier. So we switch over to the implementation of 4x4 bit Vedic multiplier which uses the

2x2 bit multiplier as a basic building block. The same method can be extended for input bits 4 & 8. But for higher no. of bits in input, little modification is required.

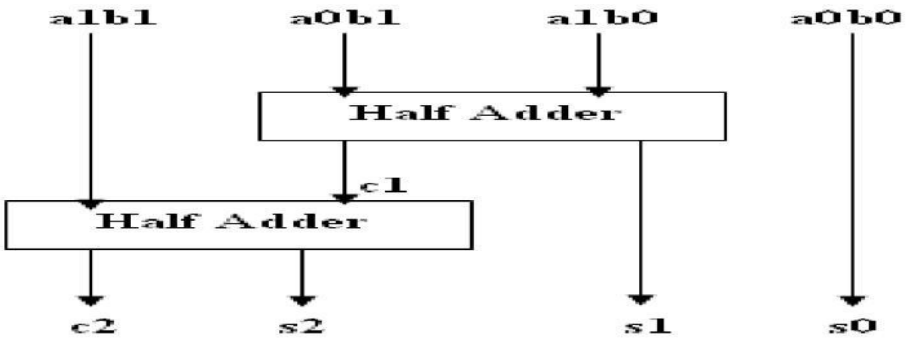

Fig.4 Block Diagram of 2x2 bit Vedic Multiplier

The 4x4 bit Vedic multiplier module is implemented using four 2x2 bit Vedic multiplier modules as discussed in Fig. 4. Let"s analyze 4x4 multiplications, say A= A3 A2 A1 A0 and B= B3 B2 B1 B0. The output line for the multiplication result is – S7S6S5S4 S3 S2 S1 S0 .Let"s divide A and B into two parts, say A3A2 & A1 A0 for A and B3 B2 & B1B0 for B. Using the fundamental of Vedic multiplication, taking two bit at a timeand using 2 bit multiplier block, we can have the following structure for multiplication as shown in Fig. 4.

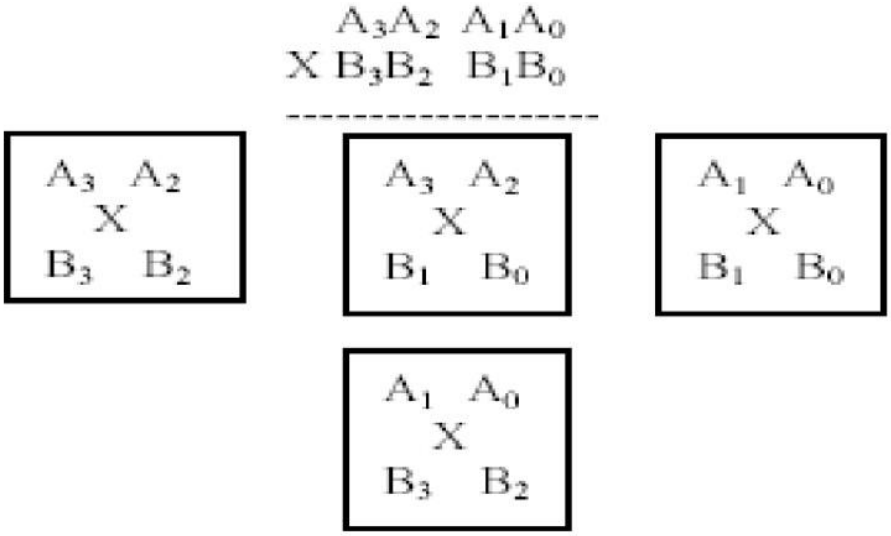

Fig.5 Sample Presentation for 4x4 bit Vedic Multiplication

Each block as shown above is 2x2 bit Vedic multiplier. First 2x2 bit multiplier inputs are A1A0 and B1B0. The last block is 2x2 bit multiplier with inputs A3 A2 and B3 B2. The middle one shows two 2x2 bit multiplier with inputs A3 A2 & B1B0 and A1A0 & B3 B2. So the final result of multiplication, which is of 8 bit, S7 S6S5S4 S3 S2 S1 S0. To understand the concept, the Block diagram of 4x4 bit Vedic multiplier is shown in Fig. 5. To get final product (S7 S6 S5 S4 S3 S2 S1 S0), four 2x2 bit Vedic multiplier (Fig. 3) and three 4-bit Ripple-Carry (RC) Adders are required. The proposed Vedic multiplier can be used to reduce delay. Early literature speaks about Vedic multipliers based on array multiplier structures. On the other hand, we proposed a new architecture, which is

efficient in terms of speed. The arrangements of RC Adders shown in Fig., helps us to reduce delay. Interestingly, 8x8 Vedic multiplier modules are implemented easily by using four 4x4 multiplier modules

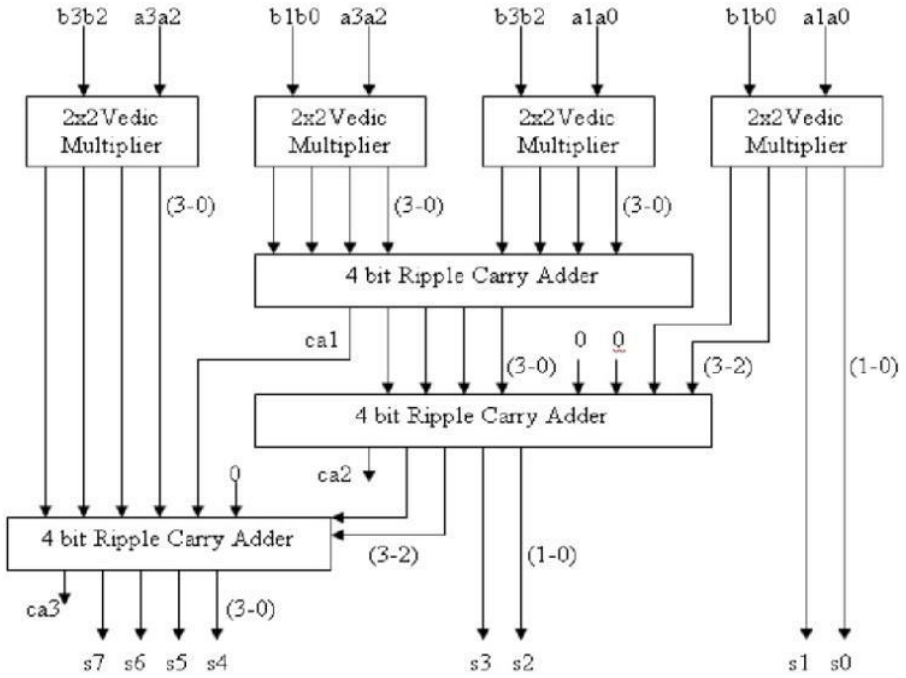

Fig.6 Block Diagram of 4x4 bit Vedic Multiplier

RESULTS:

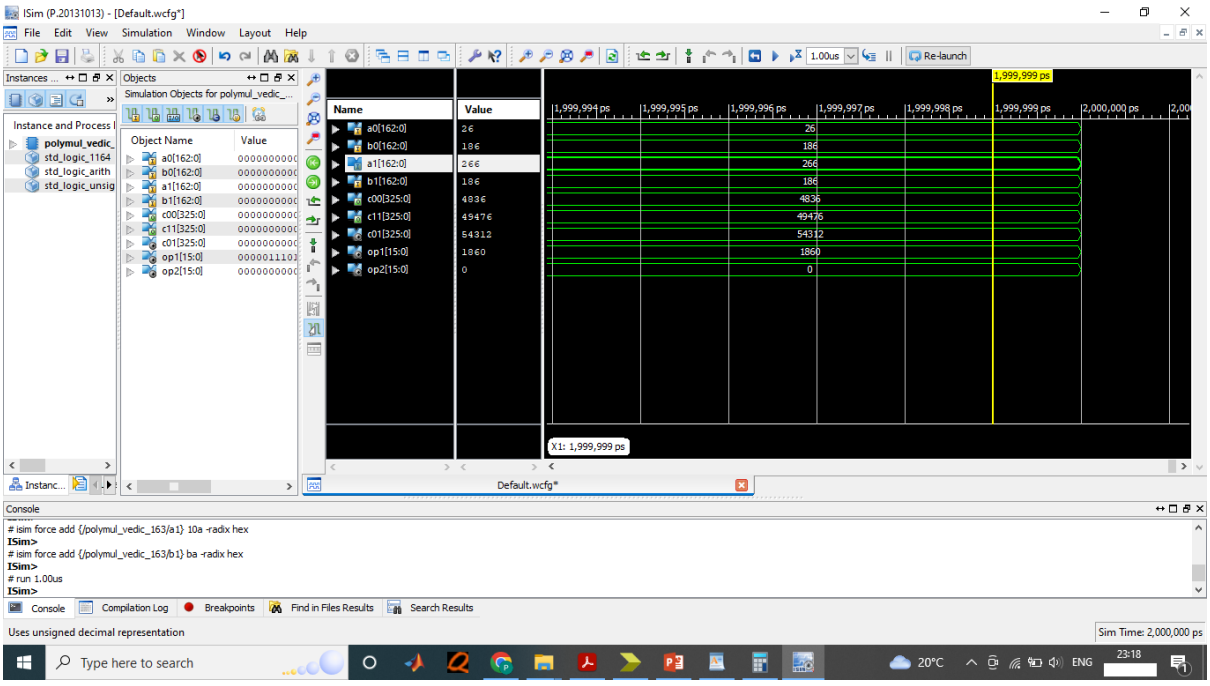

Fig7: Proposed simulation result

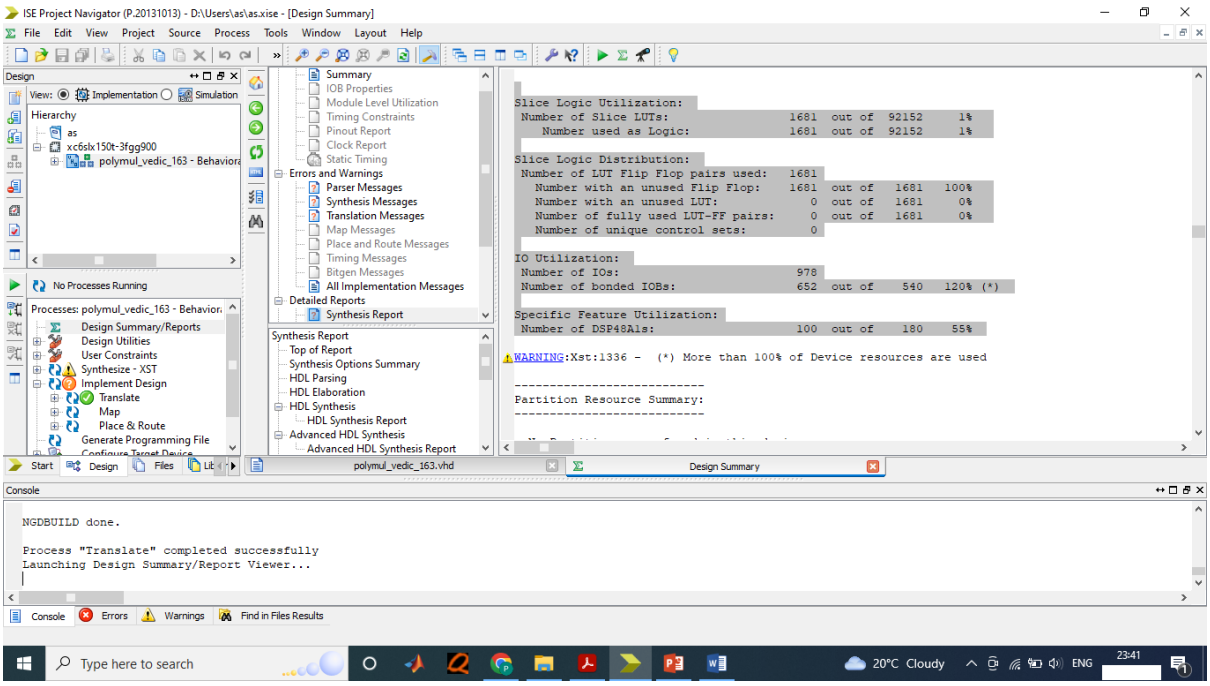

Fig8: Proposed Gate density

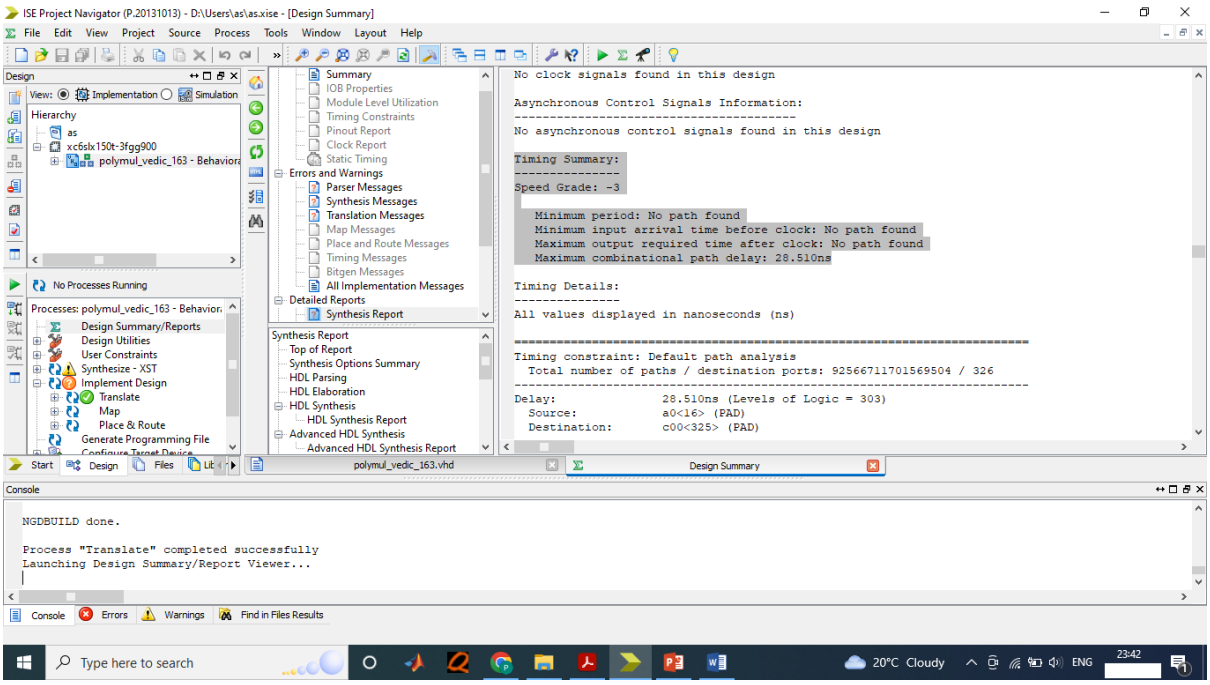

Fig: Proposes timing report

CONCLUSION:

The proposed method was implemented on FPGA for different operand sizes and its performance parameters were compared with other algorithms. Implementation results indicated that the proposed method is, on average,

faster than Karatsuba and faster than the OKA. Comparing with state-of-the-art works also indicated that the design has higher speed and lower ADP, which demonstrates the efficiency of the design.

FUTURE SCOPE:

As part of future work, the hierarchical storage format and related techniques will be applied to other linear algebra operations such as matrix factorizations (e.g. LU) that have high enough complexity to benefit from this approach. An object-oriented framework will be designed to encapsulate the ideas presented here, including polyalgorithms and storage format independence, in a format that can easily be used by numerical math libraries and applications for enhanced performance. A theoretical model that includes memory hierarchy behavior, processor architectural features and algorithmic characteristics will be devised to provide the capability to analytically determine performance costs. The analytical model would also be useful in a polyalgorithmic library for choosing the algorithm that performs best for a given set of parameters. The performance framework for writing high-performance linear algebra programs will be extended to cover parallel and distributed environments. Efficient methods for parallelizing algorithms in the hierarchical formulation will be investigated.

REFERENCES

[1] C. P. Rentería-Mejía, A. López-Parrado, J. VelascoMedina, "Hardware Design of FFT Polynomial Multipliers", 978-1-4799-2507-0/14/$31.00 ©2014 IEEE.

[2] Lo Sing Cheng, Ali Miri, Tet Hin Yeap, "EFFICIENT FPGA IMPLEMENTATION OF FFT BASED MULTIPLIERS", 0-7803-8886-0/05/$20.00 ©2005 IEEE.

[3] E. Theochari, K. Tatas, D. J. Soudris, K. Masselos, K. Potamianos, "A REUSABLE IP FFT CORE FOR DSP APPLICATIONS", 0-7803-8251-X/04/$17.00 © 2004 IEEE.

[4] A.Ronisha Prakash, S.Kirubaveni, "Performance Evaluation of FFT Processor Using Conventional and Vedic Algorithm", 2013 IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology (ICECCN 2013), 978-1-4673-5036- 5/13/$31.00 © 2013 IEEE 89. [5] Ali Chamas Al Ghouwayel, Amin Haj-Ali and Zouhair El-Bazzal, "Towards a Triple Mode Common Operator FFT for SoftWare Radio Systems", 19th International Conference on Telecommunications (ICT 2012), 978-1- 4673-0747-5/12/$31.00 ©2012 IEEE.

[6] P. D. Chidgupkar and M. T. Karad, "The Implementation of Vedic Algorithms in Digital Signal Processing", Global J. of Engg. Edu., volume 8, Issue no. 2, Year 2004.

[7] M. Moreno and Y. Xie, "FFT-based dense polynomial arithmetic on multicores", 23rd Int. conf. on high perf. Comp. systems and applic., June Year 2009, p.p. 378- 399.

[8] Sushma R. Huddar and Sudhir Rao, Kalpana M., Surabhi Mohan, "Novel High Speed Vedic Mathematics Multiplier using Compressors", 978-1-4673-5090- 7/13/$31.00 ©2013 IEEE.

[9] Laxman P.Thakre, Suresh Balpande, Umesh Akare, Sudhir Lande, "Performance Evaluation and Synthesis of Multiplier used in FFT operation using Conventional and Vedic algorithms", 978-0-7695-4246-1/10 $26.00 © 2010 IEEE.

[10] Parth Mehta, Dhanashri Gawali, "Conventional versus Vedic mathematical method for Hardware implementation of a multiplier", 978-0-7695-3915-7/09 $26.00 © 2009 IEEE.

[11] M. Ramalatha, "High Speed Energy Efficient ALU Design using Vedic Multiplication Techniques", 978-1-4244-3834-1/09/$25.00 © 2009 IEEE.

[12] Sumit Vaidya and Deepak Dandekar, "DELAY-POWER PERFORMANCE COMPARISON OF MULTIPLIERS IN VLSI CIRCUIT DESIGN", International Journal of Computer Networks & Communications (IJCNC), Volume 2, Issue no.4, July Year 2010.