LUT Optimization In Implementation Of Combinational Karatsuba Ofman On Virtex-6 FPGA

Deepak Kapoor
IIITDM Jabalpur
deepakkapoor@iiitdmj.ac.in

Rahul Yamasani
IIITDM Jabalpur
yamasani-rahul@iiitdmj.ac.in

Saket Saurav
IIITDM Jabalpur
saket@iiitdmj.ac.in

Abhishek Bajpai
Computer Division
BARC Mumbai
abbajpai@barc.gov.in

ABSTRACT

This paper discusses different approaches that allow optimizing the combinational logic used in Multipliers for Generic ECC (Elliptic Curve Cryptography) implementation in the Galois field GF(2^n) . First, a Combinational Multiplier using Karatsuba Ofman logic with 2*2 as a base multiplier has been studied. Proper utilization of Look Up Table (LUT) at base level results in effective optimization of the hardware resources.

Hence in order to optimize LUT utilization, designs for combinational logic with 3*3 base and 2*3 base have been explored, keeping the LUT structure of Virtex-6 FPGA in mind. Comparisons have shown that, 3*3 base multipliers designed using Karatsuba Ofman algorithm outperformed 2*2 and 2*3 base Multiplier in terms of resource utilization. To further maximize utilization of hardware resources, the exploration has been further carried out using Shift and Add Algorithm(SAA) and it has been found that SAA remains optimized for lower length operands. Algorithmic and platform oriented optimization results in efficient hardware implementations. The final proposed design is a Hybrid Karatsuba Algorithm, which uses SAA at lower level and at higher level uses Karatsuba Ofman Logic. Again here using 3*3 bit Multiplier with SAA configuration is better than the other two. This approach stands a step closer for efficient implementations of fast algorithm on hardware based applications, as this hybrid multiplier is found to use least number of FPGA resources.

All the operations in this paper have been performed based on Virtex-6 ML605 using ESD tool as XILINX 12.1

Keywords

Karatsuba Ofman Multiplication; FPGA; LUT; Shift and Add Algorithm; Combinational Multiplier.

SEM4HPC'16, May 31 2016, Kyoto, Japan

ACM ISBN 978-1-4503-4351-0/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2916026.2916030

1. INTRODUCTION

Multiplication unit is the critical component in ECC design. Using Classical multiplication algorithm would be costlier as it has a complexity of O (n^2). Moreover, this becomes a serious concern to the availability of hardware resources, as generic ECC implementation involves operands with lengths as large as 571 bits. Hence, optimization of Multiplication unit becomes crucial for efficient implementation of ECC.

There are several ways of implementing finite field multiplication , however Karatsuba Ofman [5] algorithm is the most efficient . Accordingly, the authors have analyzed and discussed different ways in optimizing combinational logic in multipliers with existing algorithms and came up with the designs for 2n*2n, 5n*5n and 3n*3n (where n is 2^m and m is a positive integer) Combinational Multipliers using Karatsuba Ofman algorithm with 2*2, 2*3 and 3*3 as base multipliers. Here, multipliers designed with 3*3 configuration as a base multiplier utilizes least number of resources in terms of LUT [1] utilization. In the later sections the authors explored the proposed configurations using SAA[7] and found that, SAA requires lesser number of operations than conventionally used Karatsuba Ofman Algorithm at base levels. However on extending to higher length operands, Karatsuba Ofman logic performs better than Shift and Add Algorithm. Hence, replacing the base modules with Shift and Add would utilize LUT's effectively. With this idea, the authors came up with the idea of Hybrid Karatsuba Algorithm which uses SAA at lower level and Recursive Karatsuba Algorithm at higher levels.

The authors have studied different algorithm for implementation of Karatsuba Algorithm and found [8] & [9] Debdeep Mukhopadhyay et al. to be most optimized implementation. In it they have presented a detailed study of Karatsuba multiplier for GF (2^{233}). They observed that General Karatsuba utilizes LUT to maximum extent. In addition to this, they proposed a Hybrid Karatsuba multiplier which uses General Karatsuba Multiplier at the base levels and has shown that, their proposed model utilizes least amount of resources . In [4] an analysis of high performance FPGA implementations of parallel GF (2^{233}) multipliers, namely Classical, Karatsuba, Massey-Omura [3], and Sunar-Koc [10], with respect to area and time complexity is done. Their results show that Karatsuba algorithm provides at-most optimiza-

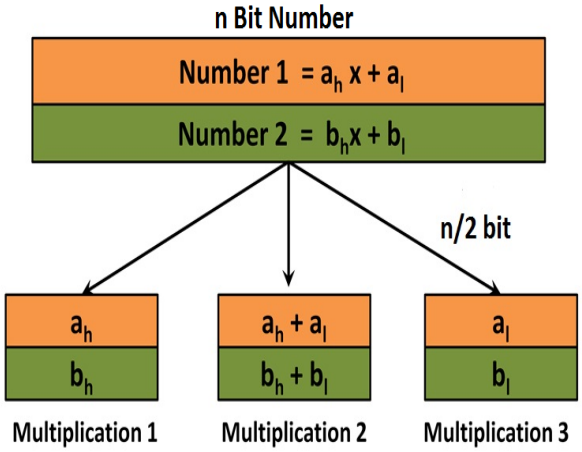

Figure 1: Karatsuba Ofman Algorithm

tion in terms of resource utilization.

The paper is structured as follows. Starting with Section 2, it summarizes various techniques of Karatsuba Ofman Algorithm. Section 3 gives a brief account on LUT's in Virtex-6 FPGA. Section 4 analyses, compares and summarizes utilization of LUT's when implemented with different variations of Karatsuba Algorithm. Section 5 includes results of the Hybrid Karatsuba with SAA. Section 6 delineates the Combinational path delays based on Virtex-6 ML605 with speed -1 [2]. The last section is the conclusion, with practical evaluation of proposed algorithm with existing models.

2. DESCRIPTION OF KARATSUBA OFMAN ALGORITHM

This section gives a brief account on Classical Karatsuba Ofman Multiplication(CKM), Recursive Karatsuba Multiplication (RKM), General Karatsuba Multiplication (GKM), Shift and Add Algorithm (SAA) and finally a Hybrid version of Karatsuba Ofman and SAA. In addition to this, it also includes the number of operations (XOR + AND) required to perform multiplication using Karatsuba Ofman Algorithm and Shift and Add Algorithm.

2.1 Classical Karatsuba Ofman Algorithm:

Karatsuba and Ofman proposed an approach which reduces the complexity involved in multiplication. Though originally developed to reduce the complexity involved in multiplying decimal numbers, the same approach can be applied to polynomial multiplication as well. The Karatsuba multiplier splits the n bit multiplicands into 2-term polynomials as shown in equation (1)

$$A(x) = (a_h)x + a_l \qquad B(x) = (b_h)x + b_l \tag{1}$$

The n Bit multiplication is done by using three n/2 bit multiplication as shown in Figure 1. Here, Karatsuba and Ofman Algorithm reduces the complexity by computation of middle term of Equation 2 as shown in Equation 3. (Multiplication is performed in binary field).

$$A(x)B(x) = (a_h b_h)x^2 + (a_h b_l + a_l b_h)x + a_l b_l \tag{2}$$

$$(a_h b_l + a_l b_h) = (a_h + a_l)(b_h + b_l) + a_h b_h + a_l b_l \tag{3}$$

On applying this logic, the final product would be as shown in Equation 4

$$C(x) = (a_h b_h)x^2 + ((a_h + a_l)(b_h + b_l) + a_h b_h + a_l b_l)x + a_l b_l \tag{4}$$

Applying Karatsuba Ofman logic reduces complexity as the number of Multiplications required are reduced to three instead of four as in general multiplication. The significant benefit of Karatsuba Ofman logic becomes evident on extending to higher levels.

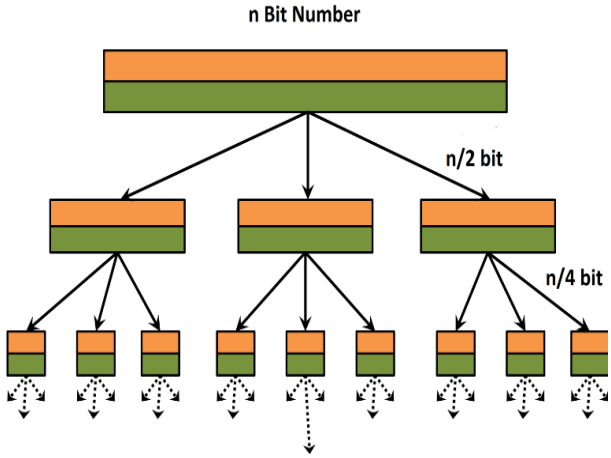

Figure 2: Recursive Karatsuba Algorithm

2.2 Recursive Karatsuba Algorithm:

Basically, Recursive Karatsuba Algorithm (RKM) works on the policy of Divide and Conquer. It splits the operands in to two halves and then CKM is applied to each chunk as shown in Figure 2. Classical Multiplication of n bit number has a complexity of O(n^2). The number of ANDs and XORs in here would be n^2 and $(n-1)^2$respectively where as the number of AND and XOR operations for multiplying two n-bit numbers for RKM is given by equations 5 and 6 respectively [8].

$$AND : 3^{log_2 n} \tag{5}$$

$$XOR : \sum_{p=0}^{\log_2 n} 3^r (\frac{4n}{2^p} - 4) \tag{6}$$

From the above equations it can be found that it has a complexity of $O(n^{1.58})$ [6] which makes it better than Classical Multiplication that has a complexity $O(n^2)$. This makes Karatsuba Ofman Algorithm an obvious choice for large operands.

2.3 General Karatsuba Multiplication:

The idea of dividing the operand into two halves as described in RKM can be generalized. This idea has given the scope of introducing a new version of Karatsuba algorithm called General Karatsuba Multiplication. The idea of General Karatsuba Multiplication(GKM) is presented in [11]. In this paper General Karatsuba Algorithm has been performed by dividing the operand into 4 chunks. Using General Karatsuba designs for 32*32 Multiplier from 8*8 instead of 16*16 .similarly 40*40 Multiplier from 10*10 and 48*48 Multiplier from 12*12 have been discussed.

2.4 Hybrid Karatsuba Multiplication:

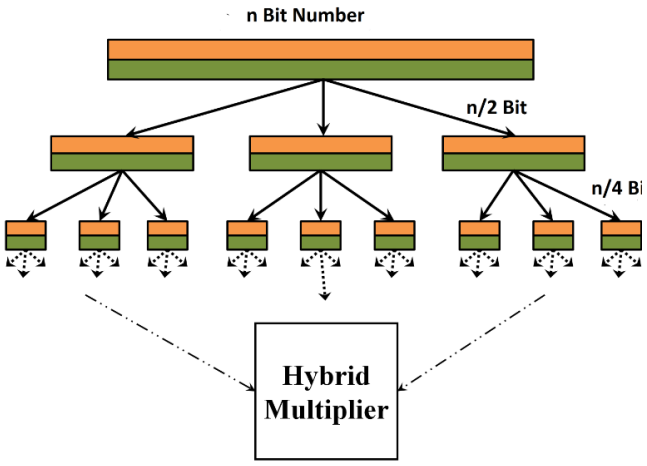

Figure 3: Hybrid Karatsuba Algorithm

Hybrid Karatsuba Multiplication [8] (HKM) is an improvement to Recursive Karatsuba Multiplication where the base level multiplier is performed with an algorithm other than Classical Karatsuba Multiplication. The hierarchy involved in HKM is shown in Figure 3 (basically in this paper the authors used Look Up Table to perform base level multiplications when Recursive Karatsuba Multiplication is performed). This paper includes a Hybrid Karatsuba Multiplication performed with SAA at the base level.

2.5 Shift and Add Algorithm:

Multiplication using Shift and Add Algorithm is synonymous to that of a School book Multiplication. In this method, multiplicand 'A'is added to itself 'B'times, where 'B'denotes the multiplier. To multiply two operands, the algorithm suggests to take the digits of the multiplier one at a time from right to left, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate positions to the left of the earlier results. Finally XOR operation is performed to get the final result. Number of XOR and AND operations required for performing n bit multiplication using SAA is demonstrated in the Equations 7 and 8.

$$AND : n^2 \tag{7}$$

$$Xor : (n-1)^2 \tag{8}$$

Table 1 compares numbers of operation (XOR + AND) required for RKM and SAA. From the comparison, it is evident that SAA requires lesser numbers of operations for lower length operands.

Table 1: Operations in RKM and SAA for 2n*2n Configuration

Number of Bits	RKM		SAA	
	XOR	AND	XOR	AND
2	4	3	1	4
4	24	9	9	16
8	100	27	49	64
16	360	81	225	256

Number of operations (XOR + AND) required for SAA is less than that of Karatsuba at lower bits. Based on the calculations made from Table 1, it can be shown that SAA at the base level is the most optimized one. These expectations matches with the results that are described in Section 4.3.

3. LUT(LOOK UP TABLE):

In Virtex-6 each of the logic function generator is associated with 6 independent inputs and 2 independent outputs. LUT can be implemented either as arbitrarily defined 6 input 1 output O_6 or 5 input 2 output(O_5&O_6) configurations, provided the two function generators share common inputs as shown in Figure 4. An optimization stands at the door, if we are able to optimize the usage of O_5 and O_6 outputs of function generators. The possibility to implement function generator as 5 input gives a scope for optimization. To understand the criterion, let's discuss the utilization of inputs and outputs in LUT with the examples shown in Equations 9 and 10.

$$Y_1 = A \oplus B \oplus C \oplus D \tag{9}$$

$$Y_2 = A_1 \odot B_1 \odot C_1 \odot D_1 \tag{10}$$

Y_1 and Y_2 are derived from different inputs. Here inputs being different, Y_1 and Y_2 are operated using two LUT's as only O_6 is used leaving shared output O_5 & O_6 unused.

$$Y_1 = A \oplus B \oplus C \oplus D \tag{11}$$

$$Y_1 = A \odot B \odot C \odot D \tag{12}$$

Now, consider two other operations as in equations 11 and 12. The difference is both the functions have same set of inputs. Here, Y_1 and Y_2 being different functions we require two outputs ($O_5 and O_6$), but inputs being same the above operations can be performed using a single LUT. From these examples, it can be said that one can optimize the resource utilization by grouping the inputs in a proper way.

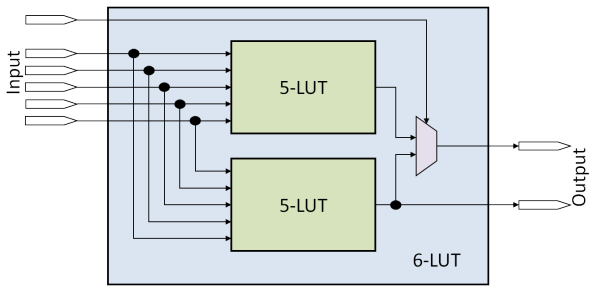

Figure 4: Structure of LUT

4. DESCRIPTION OF LUT UTILIZATION IN COMBINATIONAL MULTIPLIERS:

This section delineates the resource utilization in terms of LUTs for different versions of Karatsuba Ofman that are prescribed in section 2. It also includes LUT utilization in terms of O_6 and O_5&O_6 outputs.

4.1 Multiplication using Recursive Karatsuba Algorithm:

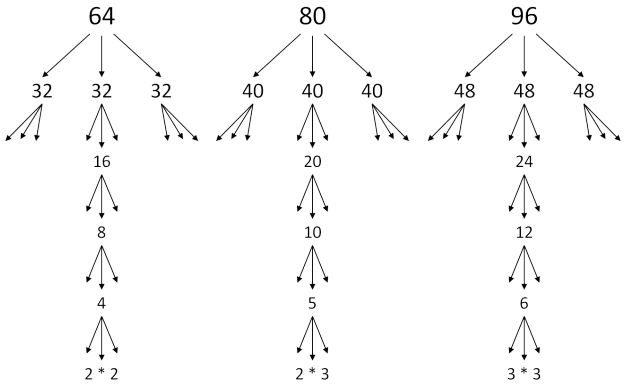

Figure 5: Hierarchy in Recursive Karatsuba Ofman Multiplier

4.1.1 *Multiplication with 2*2 bit base Multiplier*

First, a 2n*2n bit Combinational Multiplier using RKM with 2-bit multiplication table at the base level(as described in section 2.2)has been designed . The hierarchy of recursions involved in Combinational Multiplier using RKM is shown in Figure 5. The 2*2 base multiplier involves operands of size two bits. As we all know that, multiplication of two n bit numbers in binary field would result in output with maximum of 2n-1 bits. Hence 2*2 multiplication has maximum of 3 bit output which consumes two LUTs. The result containing 3 bits in maximum, it can be evident that one of the LUT is utilized incompletely(as each LUT is associated with 2 Outputs). Table 2 describes LUT utilization at each level with corresponding $O_6, O_5\&O_6$ outputs. From the Table 2 it can summarized that O_6 configuration utilizes more LUT than $O_5\&O_6$ output configuration. Since, LUT utilization is more in the unshared O_6 configuration, it can be concluded that LUT utilization is improper.Hence, with the idea to maximize the utilization of LUTs the base multiplier has been changed from 2*2 to 3*3.

Table 2: LUTs utilization in 2n*2n Bit Configuration using RKM

Multiplier	**LUTs Used**	O_6	O_5 & O_6
2*2	2	1	1
4*4	8	4	4
8*8	41	25	16
16*16	140	85	55
32*32	466	293	173
64*64	1535	1032	503

4.1.2 *Multiplication with 3*3 base Multiplier*

Similarly, a 3n*3n bit RKM Combinational Multiplier is designed with 3-bit multiplication table at the base level. In 3*3 Multiplier, the output being a 5 bit number consumes 3 LUTs. Table 3 describes LUT utilization at each level with corresponding $O_6, O_5\&O_6$ outputs. The increased trend in the shared output configuration $O_5\&O_6$ than O_6 as compared to 2n*2n Bit multiplication indicates that utilization of LUTs with 3*3 as base is more efficient. The reason being, for every 2*2 multiplier 1 LUT out of 2 (50 %) is utilized completely whereas in 3*3 base multiplier 2 LUTs out of 3(66.67 %) are completely utilized. This increased usage of LUT as compared to 2*2 indicates the merit involved in 3*3 convention.

Table 3: LUTs utilization in 3n*3n Bit Configuration using RKM

Multiplier	**LUTs Used**	O_6	O_5 & O_6
3*3	3	1	2
6*6	17	7	10
12*12	63	30	33
24*24	217	103	114
48*48	744	371	373
96*96	2382	1272	1110

4.1.3 *Multiplication with 2*3 bit Multiplier*

In order make best usage of LUTs, the base multiplier should be in such a way, that it should completely accommodate LUT outputs to a maximum extent. These can be achieved with one of the operands as a 2 bit number and the other as a 3 bit number. With this idea, a 5n*5n bit RKM Combinational Multiplier has been designed with 2*3-bit multiplication table at the base level.

Table 4: LUTs utilization in 5n*5n Bit Configuration using RKM

Multiplier	**LUTs Used**	O_6	O_5 & O_6
2*3	2	0	2
5*5	11	7	4
10*10	46	31	15
20*20	169	118	51
40*40	570	404	166
80*80	1851	1348	503

Table 4 shows that LUTs are fully utilized at lower levels. Unlikely, as we proceed to higher levels LUTs utilization in O_6 configuration is more than LUTs in $O_5\&O_6$ configuration. Numerous factors account for this behavior. The primary reason being increase in the number of XORs when carried to higher levels. Another reason is asymmetry involved in 2*3 base Multiplier. The quantitative analysis made in Figure 6 also shows that constructing combinatorial logic with 3*3 as base convention would consume least number of FPGA resources. Here the novelty lies in 3n*3n bit multiplier as the 3n*3n convention is best suited with the LUT structure of Virtex-6 FPGA.

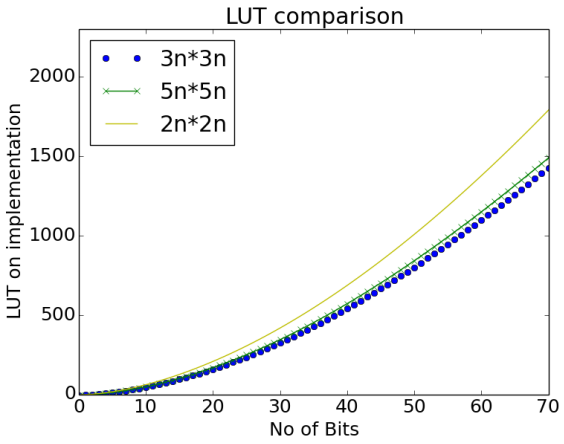

Figure 6: Comparison of LUT in 3n*3n ,2n*2n and 5n*5n Bit

4.2 Multiplication using General Karatsuba Algorithm:

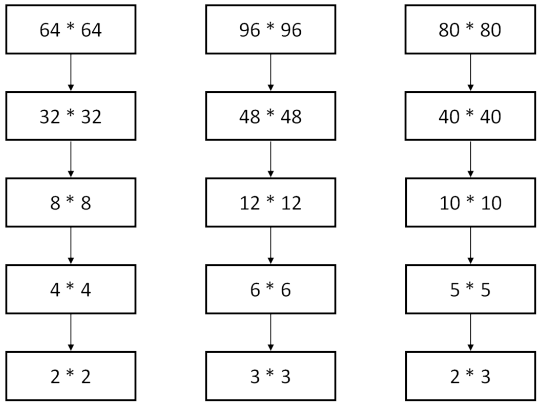

Figure 7: Hierarchy in General Karatsuba

Next modules for 2n*2n, 3n*3n and 5n*5n bit multiplication is performed using General Karatsuba Multiplication.

Table 5: LUT in GKM with base 2*2

2*2	4*4	8*8	32*32	64*64
2	8	41	445	1467

Table 6: LUT in GKM with base 3*3

3*3	6*6	12*12	48*48	96*96
3	17	63	689	2226

Table 7: LUT in GKM with base 2*3

2*3	5*5	10*10	40*40	80*80
2	11	46	511	1669

In 64 bit multiplier, a 32*32 multiplier has been designed from 8*8 bit multiplier using the GKM with 8*8 bit multiplier from RKM . LUT consumption details are incorporated in Table 5.Similarly, Table 6 and 7 shows the LUT utilization for implementing 96*96 and 80*80 Combinational Multipliers using GKM as shown in Figure 7.The comparisons has been made with HKM and RKM is shown in figure 10.

4.3 Multiplication using Shift and Add Algorithm

Here 2n*2n bit, 3n*3n bit and 5n*5n bit multipliers are implemented using Shift and Add Algorithm to find out merits in SAA implementation. Tables 8, 9 and 10 furnishes the number of LUTs used in 2n*2n, 3n*3n and 5n*5n base configurations respectively.

Figure 8 is a comparison in between SAA and RKM for 3n*3n Multiplier (as 3n*3n is most optimized with respect to others) showing that SAA remains the most optimized for operands for lower bits only as bits increases number of And and Xor operations increases dramatically making it less optimized . The data from the tables also supports the fact.

Table 8: LUT in SAA in 2n*2n configuration

2*2	4*4	8*8	16*16	32*32	64*64
2	6	26	116	485	1987

Table 9: LUT in SAA in 3n*3n configuration

3*3	6*6	12*12	24*24	48*48	96*96
3	15	62	269	1111	4515

Table 10: LUT in SAA in 5n*5n configuration

5*5	10*10	20*20	40*40	80*80
10	43	182	770	3123

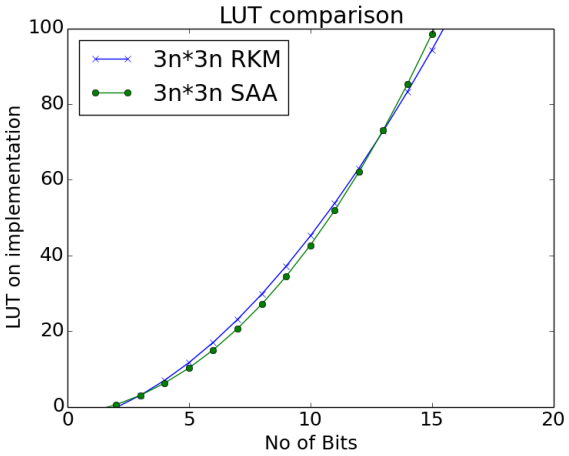

Figure 8: Quantitative comparison of SAA and RKM LUT at lower levels

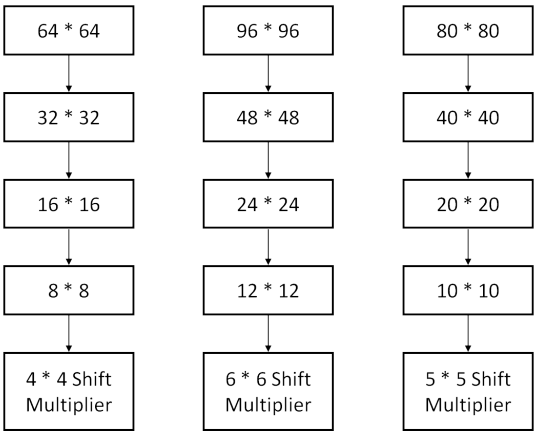

Figure 9: Hierarchy in Hybrid Karatsuba Multiplier

5. MULTIPLICATION USING HYBRID KARATSUBA ALGORITHM

In the section 4.3 it is shown that SAA is the most optimized one at lower levels. In order to confirm the observation made, the authors designed a 2n*2n bit Hybrid Karatsuba Multiplier with 4*4 Karatsuba Multiplier replaced by 4 bit SAA module. On practical evaluation, LUT utilization in HKM is most optimized. As Hybrid Multiplier involves SAA as base multiplier it requires least amount of resources (See Table 2). Results of 64 bit Hybrid Multiplier designed using 4*4 SAA module are depicted in Table 11 along with O_6 and O_5&O_6 configurations.

Table 11: LUTs utilization in 2n*2n Configuration using HKM

Multiplier	LUTs Used	O_6	O_5 & O_6
4*4	6	4	2
8*8	24	14	10
16*16	96	61	35
32*32	335	219	116
64*64	1133	790	343

Similarly, in 3n*3n bit HKM is designed using the 6 bit SAA module , while 5n*5n bit HKM is implemented using 5 bit SAA module. LUT utilization in 3n*3n Bit and 5n*5n bit HKM configurations are portrayed in Table 12 and 13.

Table 12: LUTs utilization in 3n*3n Configuration using HKM

Multiplier	LUTs Used	O_6	O_5 & O_6
6*6	15	12	3
12*12	54	41	13
24*24	194	145	49
48*48	655	488	167
96*96	2132	1619	513

Table 13: LUTs utilization in 5n*5n Configuration using HKM

Multiplier	LUTs Used	O_6	O_5 & O_6
5*5	10	7	3
10*10	38	25	13
20*20	143	101	42
40*40	493	353	140
80*80	1610	1170	740

In 3n*3n Hybrid Karatsuba Multiplier, LUT utilization in O_6 configuration exceeds than in O_5 &O_6 configuration. Even then the results obtained are better than RKM and GKM as shown in 10 because the SAA itself employs better usage of resources than Karatsuba Ofman Algorithm.

Figure 11 shows that 3n*3n bit multiplier provides the most optimized results in HKM.

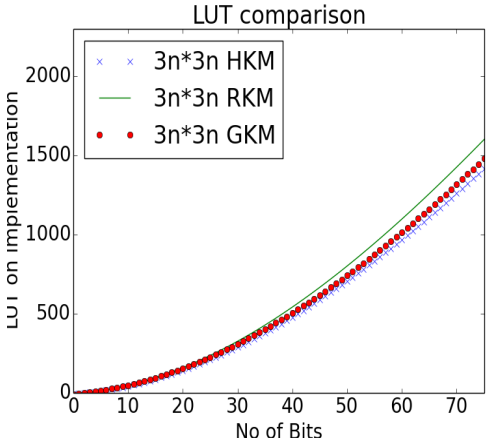

Figure 10: LUT Comparison between HKM ,GKM and RKM

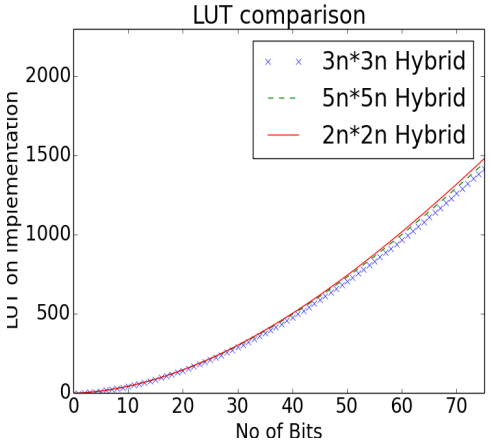

Figure 11: LUT Comparison in Hybrid Karatsuba in various configurations

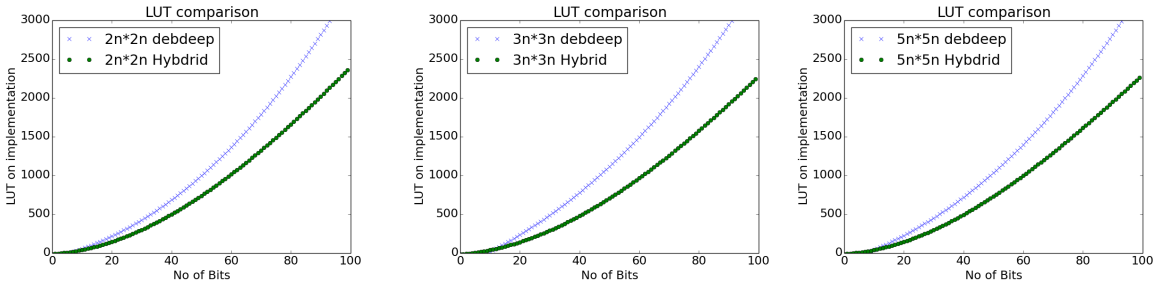

Figure 12: Comparison of proposed Hybrid SAA and Debdeep's Hybrid

6. COMBINATIONAL PATH DELAY

Combinational path delay primarily depends upon the number of recursion levels. In addition to this, the number of XORs and ANDs involved also have an effect on delay. An account of Combinational path delay obtained from various Multiplications is given in Table 14.

Table 14: Combinational path delay in nanoseconds(ns)

Type of Multiplier	96*96	80*80	64*64
General Karatsuba	6.226	6.231	6.183
Hybrid Karatsuba	6.441	6.382	6.280
Recursive Karatsuba	5.982	6.259	6.105
Shift and Add	31.921	20.657	16.774

Delay in modules using Shift and add Algorithm are maximum, as they utilize highest number of LUTs. Delays obtained in RKM and GKM are better than HKM as lower levels here involve SAA modules, that has higher delays.

7. CONCLUSION:

In this paper a Hybrid Karatsuba which uses best of Recursive Karatsuba and SAA has been proposed. The 3n*3n bit hybrid Combinational Karatsuba, with Shift and Add as base multiplier, is the most optimized in terms of LUT Utilization.

The results of Hybrid Karatsuba with Shift and Algorithm provide better results than that of Debdeep Mukhopadhyay et al. [8] & [9] . Their results describe that using General Karatsuba utilizes LUTs to a maximum extent. But on the contrary if we use Hybrid Karatsuba with Shift and Add Algorithm resource utilization is optimum. The comparison made in Figure 12 illustrates the facts.

8. REFERENCES

[1] *Virtex-6 FPGA Configurable Logic Block.* : UG346 :(V1.2), February 8, 2012.

[2] *Virtex-6 ML 605 Hardware User Guide.* UG 354 (V 1.2), Octiober 2,2012.

[3] A.Reyhani-Maoleh, and Hasan, M. A. A new construction of massey-omura parallel multiplier over GF(2^m). *IEEE Trans .Computers Pp.511-520* (MAY 2002).

[4] Grabbe, C., et al. FPGA designs of parallel high performance $GF(2^{233})$ multipliers power attack resistant efficient fpga architecture for karatsuba multiplier. *IEEE International Symposium on Circuits and Systems (ISCAS 03) II* (2003).

[5] Karatsuba, A., and Ofman, Y. Multiplication of multi digit numbers on automata. *Soviet Physics Doklady 7* (1963).

[6] Montgomery, P. L. Five, six, and seven-term karatsuba-like formulae. *IEEE Transactions On Computers 54*, 3 (MARCH 2005).

[7] N.Marimuthu, C., Thangaraj, D. P., and Ramesan, A. Low power shift and add multiplier design. *International Journal of Computer Science and Information Technology II*, 3 (JUNE 2010).

[8] Rebeiro, C., and Mukhopadhyay, D. Hybrid masked karatsuba multiplier for $GF(2^{233})$. *IEEE Transactions On Computers* (JULY 2007).

[9] Rebeiro, C., and Mukhpodhyay, D. Power attack resistant efficient fpga architecture for karatsuba multiplier. *VLSI Design, International Conference on VLSI Design 0* (2008), 706–711.

[10] Sunar, B., and Koc, C. K. An efficient optimal normal basis type ii multiplier. *IEEE Transactions On Computers 50*, 1 (JANUARY 2001).

[11] Weimerskirch, A., and Paar, C. Generalizations of the karatsuba algorithm for efficient implementations. 2006 ,http://eprint.iacr.org/2006/224.pdf.