FPGA Based Modified Karatsuba Multiplier

Jagannath Samanta
Dept. of ECE
Haldia Institute of Technology
Haldia, India
jagannath19060@gmail.com

Razia Sultana
Dept. of ECE
Haldia Institute of Technology
Haldia, India
razia04@gmail.com

Jaydeb Bhaumik
Dept. ECE
Haldia Institute of Technology
Haldia, India
bhaumik.jaydeb@gmail.com

Abstract— Finite field arithmetic is becoming increasingly a very prominent solution for calculations in many applications. In this paper, complexity and delay of six different multipliers (Mastrovito multiplier, Paar-Roelse multiplier, Massey- Omura multiplier, Hasan-Masoleh multiplier, Berlekamp multiplier and Karatsuba multiplier) are compared. Also this paper presents a modified multiplier based on Karatsuba multiplication algorithm. To optimize the Karatsuba multiplication algorithm, the product terms are splited into two alternative forms and all the terms are expressed in the repeated fashion. This Modified architecture saves the 14.9% computation time and it consumes 45.5% less slices than existing Karatsuba multiplier. The proposed architecture has been simulated and synthesized by Xilinx ISE design suite for Spartan & Vertex device family. The new architecture is simple & easy. The proposed Modified Karatsuba Multiplier (MKM) is also applied to compute the circular convolution for DSP application. In Spartan3E FPGA device family, computation of 8-bit circular convolution using Modified Karatsuba Algorithm (MKA) is 26.5% faster than Karatsuba Algorithm (KA). It also consumes 61.7% less slices than existing KA based Convolution.

Keywords- Karatsuba Algorithm; Finite fields; FPGA; VLSI; polynomial multiplication; Cicular Convolution;

I. INTRODUCTION

Galois fields have gained wide spread applications in error correcting codes and cryptographic algorithms. Further applications may be found in signal processing and pseudo random number generation. Modern applications in many cases call for VLSI implementations of the arithmetic modules in order to satisfy the high speed requirements. VLSI allows the designers to allocate complex systems consisting of several thousand or even millions transistors on one or very few chips. VLSI modules having Galois field multiplier can be classified into three categories: bit- serial multipliers [6], bit- parallel multipliers, and hybrid multiplier. Bit parallel architectures tend to be faster and only use combinatorial logic [5]. On the other hand, bit serial architectures require less area and uses registers in addition to combinatorial logic, and the hybrid multipliers, which are partially bit-serial and partially bit-parallel. Hybrid multipliers are faster than bit-serial ones, while their area is smaller than that of bit- parallel. For efficient VLSI implementation suitable hardware architecture is needed. It is obtained by using addition, multiplication, field operations, suitably in the architecture. Addition can be implemented with a very low space complexity, multiplication is required to be fast but it is implemented with a higher complexity. Efficient architectures require low complexity and fast multipliers. Assuming a basis representation of the field elements addition is a relatively inexpensive operation, whereas the other field operation, is costly in terms of gate count and delay.

In the polynomial multiplication, Karatsuba algorithm is used to make multiplication efficient which means algorithm saves multiplication at the cost of extra addition. Because multiplication is more costly than addition. Addition of two m-bit numbers require *m* no. of XOR gates. Koc *et al.* [8] have proposed a recursive algorithm for fast multiplication of large integers having a precision of 2^k computer words, where k is an integer. Their algorithm has been derived from the Karatsuba-Ofman algorithm and has the same asymptotic complexity. They have claimed that the running time of their algorithm is a little better that makes one third as many recursive calls. Murat Cenk *et al.* [9] gave improved formulas to multiply polynomials of small degree over F_2 using Chinese Remainder Theorem (CRT) that improve multiplication complexity. Gang Zhou *et al.* have presented complexity analysis and efficient FPGA (Field Programmable Gate Array) implementations of bit parallel mixed Karatsuba–Ofman multipliers in [10]. By introducing the common expression sharing and the complexity analysis on odd-term polynomials, they have achieved a lower gate bound than previous ASIC implementation. They have extended the analysis by using 4-input/6-input lookup tables (LUT) on FPGAs. They have evaluated the LUT complexity and area-time product tradeoffs on FPGAs with different computer-aided design (CAD) tools. They claim that their bit parallel multipliers consume the least resources among known FPGA implementations.

In this paper, a modified multiplier based on Karatsuba multiplication algorithm is proposed. To optimize the Karatsuba multiplication algorithm, the product terms are splited into two alternative forms and computed all the terms in the repeated fashion. This modified architecture saves the 14.9% computation time and it consumes 45.5% less slices than existing Karatsuba multiplier. The proposed design has been simulated and synthesized using Xilinx FPGA based Spartan and Vertex device family. The new architecture is simple and easy. It is also applied to compute circular convolution. In Spartan3E FPGA device family, computation of 8-bit circular convolution using MKA is 26.5% faster than

KA. It also consumes 61.7% less slices than existing KA based convolution.

The rest of the paper is organized as follows. Basics of Galois Field arithmetic and comparison of the different GF multipliers are presented in section-II. A new method for implementations of Karatsuba multipliers has been proposed in Section-III. Results & discussion are provided in Section-IV. Section-V describes application of proposed algorithm to compute the circular convolution and finally the paper is concluded in Section-VI.

II. GALOIS FIELD ARITHMETICS

Galois field defines as $GF(p^m)$ which is a field with p^m numbers of elements (p is a prime number) [7]. Furthermore, order of Galois field is the number of elements in the Galois field. Addition and multiplication are two basic operations mainly done in Galois field arithmetic. Addition and subtraction of elements of GF(2^m) are simple XOR operations of the two operands. Each of the elements in the GF is first represented as a corresponding polynomial. Multiplication operation over the Galois field is a more complex operation than the addition operation. For m=4, the product is represented as follows:

$$A(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \quad (1)$$

$$B(x) = b_3x^3 + b_2x^2 + b_1x + b_0 \quad (2)$$

$$A(x)\times B(x) = (a_3x^3 + a_2x^2 + a_1x + a_0) \times (b_3x^3 + b_2x^2 + b_1x + b_0)$$

$$= (a_3b_3)x^6 + (a_3b_2 + a_2b_3)x^5 + (a_3b_1 + a_2b_2 + a_1b_3)x^4 + (a_3b_0 + a_0b_3 + a_2b_1 + a_1b_2)x^3 + (a_2b_0 + a_1b_1 + a_0b_2)x^2 + (a_1b_0 + a_0b_1)x^1 + (a_0b_0)x^0$$

The result has seven coefficients which must convert back into a 4-tuple to achieve closure. This can be done by substituting the value of x^6, x^5 and x^4 with their polynomial representations and summing terms.

$A(x) \times B(x) = (a_3b_3 + a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3)\, x^3 + (a_3b_3 + a_3b_2 + a_2b_3 + a_2b_0 + a_1b_1 + a_0b_2)\, x^2 + (a_3b_2 + a_2b_3 + a_3 b_1 + a_2b_2 + a_1b_3 + a_1b_0 + a_0b_1)\, x + (a_3b_1 + a_2 b_2 + a_1b_3 + a_0b_0)$. (3)

Eqn. (3) is often expressed in matrix form.

$$\begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_3 + a_0 & a_3 + a_2 & a_1 + a_2 \\ a_2 & a_1 & a_3 + a_0 & a_3 + a_2 \\ a_3 & a_2 & a_1 & a_3 + a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad (4)$$

The multiplication results in eqn.(3) can be implemented as logical ANDs and the additions as logical XORs. Thus, the expression requires only 16 AND and 15 XOR to implement.

GF multipliers are dependent on addition and multiplication. Addition is easy and it equates to a bit-wise XOR of the m-tuple and is realized by an array of mXOR gates. The GF multiplier much more complicated and is the key to developing efficient of GF field computational circuits. In this section, we have conducted an extensive survey on six different GF multipliers i.e. Mastrovito multiplier, Paar-Roelse multiplier, Massey-Omura multiplier, Hasan-Masoleh multiplier, Berlekamp multiplier and Karatsuba multiplier. Six different GF multipliers are compared with their device utilization and combinational path delay using Xilinx based simulation tools on FPGA platforms. We have used the Verilog HDL language to code the all design.

Karatsuba Multiplier (KM)

In this section, we introduce the fundamental Karatsuba algorithm which can successfully be applied to polynomial multiplication. The Karatsuba Algorithm was introduced by Karatsuba in 1962. The fundamental Karatsuba multiplication for polynomial in $GF(2^m)$ is a recursive divide-and-conquer technique. It is considered as one of the fastest way to multiply long numbers. For polynomial multiplication with original Karatsuba method both operands have to be divided into two equal parts. Then each sub operands is divided again into two parts. The process will continue until this become single. Figure1 shows the block diagram of Karatsuba multiplier for degree-3 polynomials. Then we get the followings by splitting the polynomials using KM:

If $A(x)$ and $B(x)$ are field polynomials with degrees 3 over a field GF (2^4).

With the auxiliary variables

$D_0 = a_0b_0$, $D_1 = a_1b_1$

$D_2 = a_2b_2$, $D_3 = a_3b_3$

$D_{0,1} = (a_0 + a_1)\,(b_0 + b_1)$

$D_{0,2} = (a_0 + a_2)\,(b_0 + b_2)$

$D_{1,3} = (a_1 + a_3)\,(b_1 + b_3)$

$D_{3,2} = (a_3 + a_2)\,(b_3 + b_2)$

$D_{0,1,2,3} = (a_0 + a_1 + a_2 + a_3)\,(b_0 + b_1 + b_2 + b_3)$

Field multiplication can be performed into two steps. Firstly, we perform an ordinary polynomial multiplication of two field elements. Secondly, a reduction operation with an irreducible polynomial is need to be performed in order to obtain the (m - 1) degree polynomial. It is noticed that once the irreducible polynomial $p(x) = x^4 + x + 1$ has been selected, the reduction step can be accomplished by using XOR gates only [9]. From the irreducible polynomial $p(x)$ we can replace $x^4 = x + 1$, $x^5 = x^2 + x$ and $x^6 = x^3 + x^2$ to obtain $C'(x)$ as follows:

$$C'(x) = A(x)\, B(x) \bmod p(x)$$

$$C'(x) = (D_{0,1,2,3} - D_{1,3} - D_{2,0} - D_{3,2} - D_{0,1} + D_0 + D_1 + D_2)x^3 + (D_{0,2} + D_{3,2} + D_1 - D_0)\, x^2 + (D_{0,1} + D_{1,3} + D_{3,2} - D_0)x + (D_{1,3} - D_1 - D_3 + D_2 + D_0) \quad (5)$$

III. Modified Karatsuba Multiplier (MKM)

In this section our Modified Karatsuba Algorithm (MKA) has been discussed. In MKA all techniques are same as fundamental basic Karatsuba multiplier except the splitting techniques. To optimize the Karatsuba Multiplication Algorithm, the product terms are splited into two alternative forms. This reduction technique requires small area and less delay than others existing multiplication algorithms. The results are compared by using Xilinx based synthesis tools on different FPGA device family like Spartan & Vertex. Our synthesis results are better than existing basic Karatsuba algorithm which is shown in the following section. Assume

$A(x)$ and $B(x)$ are two field polynomials with degree 3 in $GF(2^4)$.

$$A(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0$$
$$B(x) = b_3 x^3 + b_2 x^2 + b_1 x + b_0$$

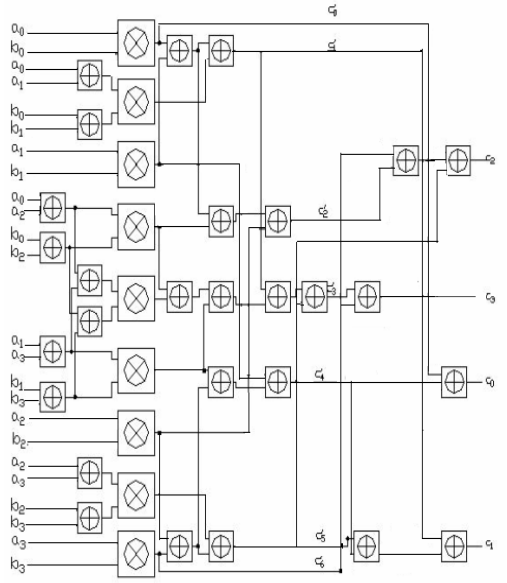


Fig.1: Block diagram of Karatsuba multiplier for degree-3 polynomials

Then we get the following expression by splitting the coefficients of $C(x)= A(x)B(x)$ polynomial using MKA.

$D_0 = a_0 b_0$, $D_1 = a_1 b_1$
$D_2 = a_2 b_2$, $D_3 = a_3 b_3$
$D_{3,2}=(a_3+a_2)(b_3+b_2)$
$D_{3,1}=(a_3+a_1)(b_3+b_1)$
$D_{3,0}=(a_3+a_0)(b_3+b_0)$
$D_{1,2}=(a_1+a_2)(b_1+b_2)$
$D_{0,2}=(a_0+a_2)(b_0+b_2)$
$D_{0,1}=(a_0+a_1)(b_0+b_1)$

Here operands are splited into two alternative terms. Employing auxiliary variables, we can obtain the following expression.

$$C(x) = D_3 x^6 + (D_{3,2} - D_2 - D_3) x^5 + (D_{1,3} - D_1 - D_3 + D_2) x^4 + (D_{0,3} - D_0 - D_3 + D_{1,2} - D_1 - D_2) x^3 + D_{0,2} - D_2 - D_0 + D_1) x^2 + (D_{0,1} - D_1 - D_0) x + D_0 \quad (6)$$

Then $C'(x)$ is computed by using the relationship $C'(x)=C(x)$ mod $p(x)$. Using the irreducible polynomial $p(x)=x^4+x+1$, terms x^4 , x^5 and x^6 in $C(x)$ are replaced by $(x+1)$, (x^2+x) , (x^3+x^2) respectively. The simplified expression of $C'(x)$ is as follows:

$$C'(x)=(D_{0,3}-D_0+D_{1,2}-D_1-D_2)x^3+(D_{0,2}+D_{3,2}+D_1-D_0)x^2+(D_{0,1}+D_{1,3}+D_{3,2}-D_0)x+(D_{1,3}-D_1-D_3+D_2+D_0) \quad (7)$$

Figure2 shows the block diagram of Modified Karatsuba multiplier for degree-3 polynomials.

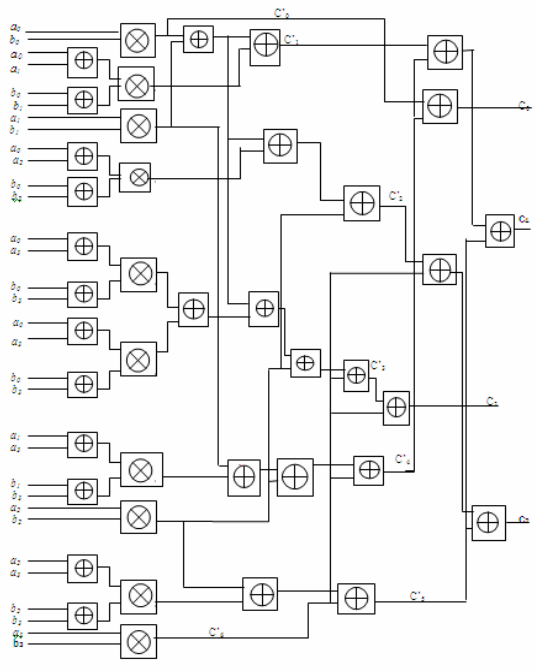


Fig.2: Block diagram of Modified Karatsuba multiplier for degree-3 polynomial

IV. RESULTS & DISCUSSION:

We have studied the performance of each multiplier over GF(2^4) employing the Xilinx ISE simulation tool. Multipliers are implemented on Spartan3E xc3s100e-4 device. These multipliers are compared based on number of slices, number of 4-input LUTs, bonded I/O blocks and delay.

TABLE 1: Comparison of different multipliers in $GF(2^4)$ field

Different GF Multipliers	No. of slices (out of 960)	No. of 4 i/p LUTs (out of 1920)	No. of bonded IOBs (out of 66)	Max. combinational path delay (ns)
Mastrovito[2]	7	12	12	13.195
Paar – Roelse [3]	7	12	12	13.083
Massy Omura [4]	7	13	12	14.932
Hasan Masoleh [5]	7	12	12	13.271
Berlekamp [6]	8	15	12	12.985
Karatsuba Multiplier (KM) [7]	9	15	12	14.790
Modified Karatsuba Multiplier (MKM)	6	11	12	13.057

Table-1 shows the result of device utilization and combinational path delay of various types of GF(2^4) multipliers. Proposed multiplier has less hardware complexity than other GF multiplier. It is also faster than other multipliers except Berlekamp.

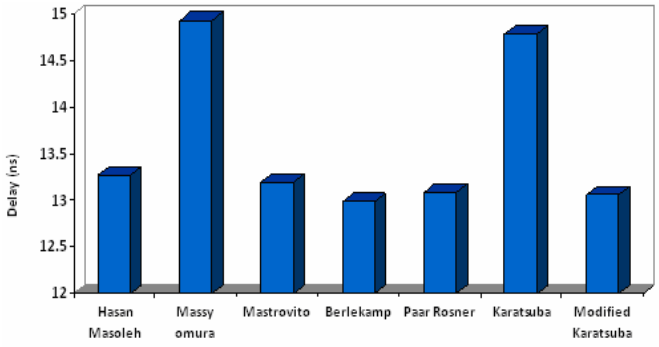


Fig. 3: Time delay graph of various multipliers in $GF(2^4)$

Figure 3 shows the delay graph of various type of finite field multiplier. From the table1, it is observed that the Berlekamp Multiplier has the lowest combinational path delay than other finite field multipliers. Highest path delay is found in Massy-Omura multiplier.

TABLE 2: Complexity comparison between KM and MKM for different GF field

	KM		MKM	
m	**# MUL**	**# ADD**	**# MUL**	**# ADD**
2	3	4	3	4
3	6	13	6	12
4	9	24	10	23
8	36	139	36	109

TABLE 3: Comparison of resource utilization between KM and MKM in $GF(2^8)$ for different Xilinx FPGA Devices.

Devices	Algo	# Slices m out of n (m/n)	# 4-i/p LUTs m out of n (m/n)	# Bonded IOB m out of n (m/n)	Delay (ns)
Spartan2 (xc2s15)	KM	66 /192	115/ 384	24/90	19.835
	MKM	36/192	62/ 384	24/ 90	15.857
Spartan 2E (xc2s50e)	KM	66 / 768	115/1536	24/182	19.095
	MKM	36 /768	62/1536	24/182	15.279
Spartan 3 (xc3s50)	KM	66/768	115/1536	24/63	16.206
	MKM	36/768	62/1536	24/63	13.948
Spartan 3E (xc3s100e)	KM	66/ 960	115/1920	24/66	20.028
	MKM	36/ 960	62/1920	24/ 66	17.035
Virtex (xcv50)	KM	66/768	115/1536	24/184	24.699
	MKM	36/768	62/1536	24/184	19.703
Virtex2 (xc2v50)	KM	66/256	115/512	24/ 88	14.759
	MKM	36/ 256	62/512	24/88	12.601
Virtex2P (xc2vp2)	KM	66/1408	115/2816	24/140	9.14
	MKM	36/1408	62/2816	24/140	7.754
Virtex4 (xc4vFx12)	KM	66/5472	115/10944	24/240	8.311
	MKM	36/5472	62/10944	24/240	7.199
Virtex E (xcv50e)	KM	66/768	115/1536	24/98	16.659
	MKM	36/768	62/1536	24/98	13.041

Table-2 shows the complexity of KM and MKM for m= 2, 3, 4 and 8. For m=4, KM requires 24 additions and 9 multiplications to compute $C(x)$ whereas MKA requires 10 multiplications and 23 additions, thus we save 1 addition. And for m=8, KM requires 139 additions and 36 multiplications to compute $C(x)$ whereas modified KM, MKA needs 36 multiplications and 109 additions. Thus MKA can save 30 additions. Table 3, compares between Karatsuba multiplier (KM) and Modified Karatsuba Multiplier (MKM) in $GF(2^8)$ field based on different Spartan & Vertex FPGA device family.

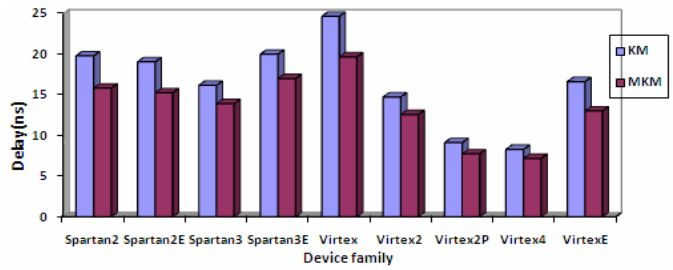


Fig.4: Delay graph of 8×8 KM and MKM on different FPGA devices

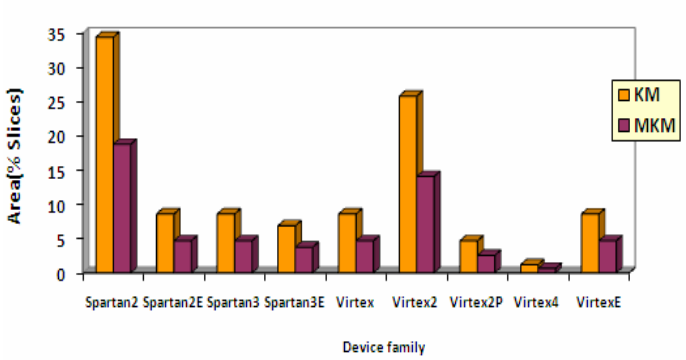


Fig. 5: Area occupied (% slices) of 8×8 KM and MKM on different FPGA devices.

Figure 4 shows the multiplication time delay of the MKM in comparison with KM for different FPGA device. The proposed architecture has very small multiplication time delay and device utilization in comparison with the other architectures. Figure 5 shows resource utilization in terms of (% of slices) necessary for the implementation. In Spartan3E, our modified Karatsuba multiplier is 14.9% faster than Karatsuba multiplier. It also consumes 45.5% less slices than KM.

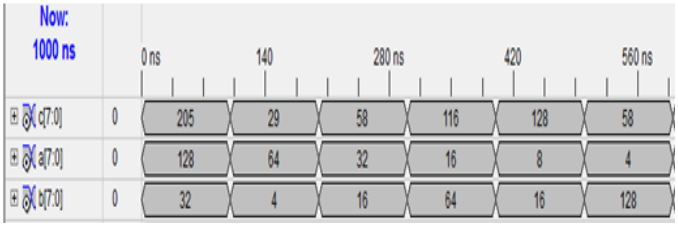
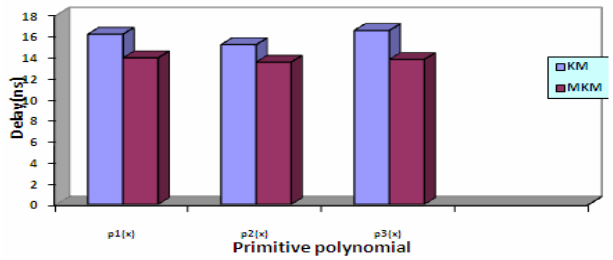


Fig. 6: Simulation results of Modified Karatsuba Multiplier

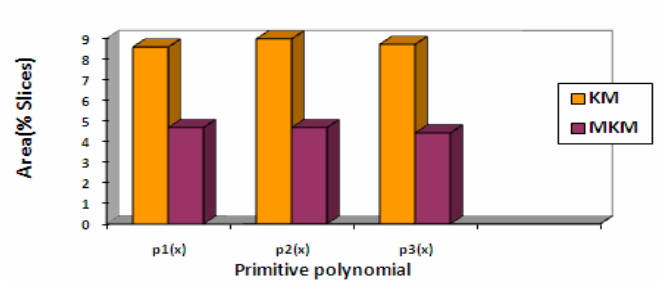
The simulation results of 8×8 MKM have been shown in Fig. 6. Figure shows the decimal equivalent of multiplication of two 8-bit numbers to give the result. Ports 'a' and 'b' are the two input ports that accept the numbers to be multiplied while the port 'c' is the output port where the product of the two aforesaid numbers is obtained.

TABLE 4: Comparison of device utilization and combinational path delay of 8×8 KM and MKM using different primitive polynomial.

Algo	# Slices (out of 768)	# 4-i/p LUT (out of 1536)	# Bonded IOB (out 63)	Delay (ns)
$p_1(x)=x^8+x^4+x^3+x^2+1$				
KM	66	115	24	16.206
MKM	36	62	24	13.948
$P_2(x)=x^8+x^5+x^3+x^2+1$				
KM	69	121	24	15.206
MKM	36	62	24	13.539
$P_3(x)=x^8+x^5+x^3+x+1$				
KM	67	116	24	16.553
MKM	34	59	24	13.798



(a)



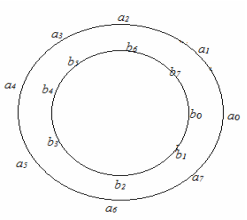
(b)

Fig7. (a) Delay (ns); (b) Area occupied (%Slices) using different primitive polynomials

Table 4 shows the simulation results for device utilization and combinational path delay of 8×8 KM and MKM using three different primitive polynomials. The multipliers are implemented on the Xilinx Spartan3 xc3s50e-4 FPGA device. Figure 7(a) shows the delay graph of KM and MKM for three types of primitive polynomial. Figure 7(b) shows the area performances of KM and MKM for three different primitive polynomials, which are given in terms of total numbers of slices necessary for the implementation. From Table 4, it is observed that in the three cases the MKM requires lesser number of slices and at the same time minimum critical path delay.

V. APPLICATION

In this Section, computation of circular convolution by employing proposed Modified Karatsuba Algorithm is presented. Assume A and B are the two sequences, where $A=\{a_0,a_1,a_2,a_3,a_4,a_5,a_6,a_7\}$ and $B=\{b_0,b_1,b_2,b_3,b_4,b_5,b_6,b_7\}$. All the points of A are placed on the outer circle in the counter clockwise direction. Starting at the same point as A, all points of B are placed on the inner circle in clockwise direction.



Expression of d_0 is obtained by multiplying the corresponding samples points and then adding the product terms.

$$d_0=a_0b_0+a_7b_1+a_6b_2+a_5b_3+a_4b_4+a_3b_5+a_2b_6+a_1b_7 \quad (8)$$

Applying Modified Karatsuba Algorithm (MKA) in equation (8) we can obtain,

$$d_0=a_0b_0+(a_7+a_1)(b_7+b_1)+a_7b_7+a_1b_1+(a_5+a_3)(b_5+b_3)+ a_5b_5+a_3b_3+(a_2+a_6)(b_2+b_6)+a_2b_2+a_6b_6+a_4b_4 \quad (9)$$

Similarly the expressions of $d_1,d_2,d_3,\ d_4\ d_5,d_6$ and d_7 are obtained and they are as follows:

$$d_1=a_0b_1+a_1b_0+a_2b_7+a_3b_6+a_4b_5+a_5b_4+a_6b_3+a_7b_2 =(a_0+a_1)(b_0+b_1)+a_0b_0+a_1b_1+(a_2+a_7)(b_2+b_7)+a_2b_2+a_7b_7+ (a_3+a_6)(b_3+b_6)+a_3b_3+a_6b_6+(a_5+a_4)(b_5+b_4)+a_5b_5+a_4b_4 \quad (10)$$

$$d_2=a_0b_2+a_1b_1+a_2b_0+a_3b_7+a_4b_6+a_5b_5+a_6b_4+a_7b_3 =a_1b_1+(a_0+a_2)(b_0+b_2)+a_0b_0+a_2b_2+(a_7+a_3)(b_7+b_3)+a_7b_7+a_3b_3 +(a_4+a_6)(b_4+b_6)+a_4b_4+a_6b_6+a_5b_5 \quad (11)$$

$$d_3=a_0b_3+a_1b_2+a_2b_1+a_3b_0+a_4b_7+a_5b_6+a_6b_5+a_7b_4 = (a_0+a_3)(b_0+b_3)+a_0b_0+a_3b_3+(a_1+a_2)(b_1+b_2)+a_1b_1+a_2b_2 +(a_4+a_7)(b_4+b_7)+a_4b_4+a_7b_7+(a_6+a_5)(b_6+b_5)+a_5b_5+a_6b_6 \quad (12)$$

$$d_4= a_0b_4+a_1b_3+a_2b_2+a_3b_1+a_4b_0+a_5b_7+a_6b_6+a_7b_5 =(a_0+a_4)(b_0+b_4)+a_0b_0+a_4b_4+(a_1+a_3)(b_1+b_3)+a_1b_1+a_3b_3+ (a_5+a_7)(b_5+b_7)+a_5b_5+a_7b_7+ a_2b_2+a_6b_6 \quad (13)$$

$$d_5=a_0b_5+a_1b_4+a_2b_3+a_3b_2+a_4b_1+a_5b_0+a_6b_7+a_7b_6 =(a_0+a_5)(b_0+b_5)+a_0b_0+a_5b_5+(a_1+a_4)(b_1+b_4)+a_1b_1+a_4b_4 +(a_6+a_7)(b_6+b_7)+a_6b_6+a_7b_7+(a_2+a_3)(b_2+b_3)+ a_2b_2+a_3b_3 \quad (14)$$

$$d_6= a_0b_6+a_1b_5+a_2 b_4+a_3b_3+a_4b_2+a_5b_1+a_6 b_0+a_7b_7 =(a_0+a_6)(b_0+b_6)+a_0b_0+a_6b_6+(a_1+a_5)(b_1+b_5)+a_1b_1+a_5b_5+ (a_2+a_4)(b_2+b_4)+a_2b_2+a_4b_4+a_7b_7+a_3b_3 \quad (15)$$

$$d_7=a_0b_7+a_1b_6+a_2b_5+a_3b_4+a_4b_3+a_5b_2+a_6b_1+a_7b_0 =(a_0+a_7)(b_0+b_7)+a_0b_0+a_7b_7+(a_1+a_6)(b_1+b_6)+a_1b_1+a_6b_6 +(a_2+a_5)(b_2+b_5)+a_2b_2+a_5b_5+(a_3+a_4)(b_3+b_4)+a_4b_4+a_3b_3 \quad (16)$$

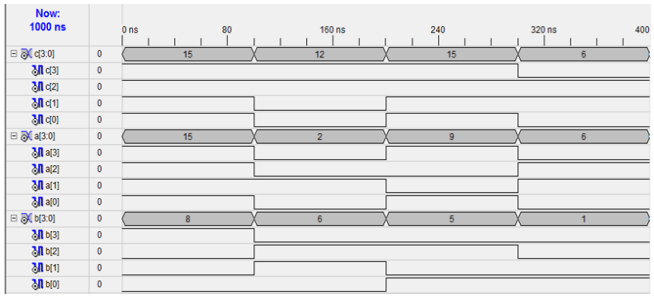


Fig. 8: Simulation result of circular convolution using MKA

TABLE 5: Comparison of device utilization and combinational path delay to compute circular convolution using KA and MKA.

Length	Algorithm	# Slices (out of 960)	# 4-i/p LUT (out of 1920)	# Bonded IOB (out of 66)	Delay (ns)
	circular convolution using KA	10	17	12	16.949
4-bit	circular convolution using MKA	7	12	12	**11.324**
	circular convolution using KA	68	118	24	18.469
8-bit	circular convolution using MKA	26	45	25	**13.567**

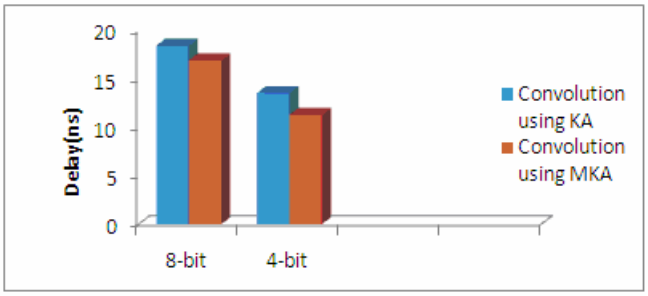


Fig. 9: Delay for comparing circular convolution using KA and MKA

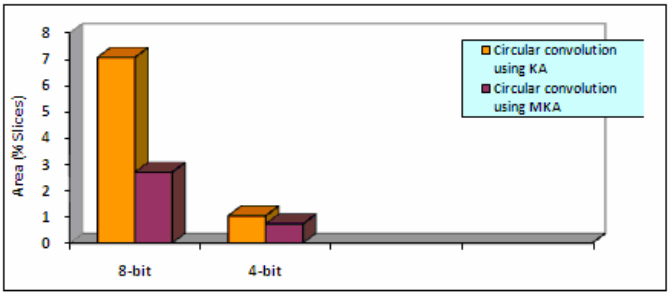


Fig. 10: Area occupied (% slices) between circular Convolution using KA and MKA

The circular convolution algorithm is coded using Verilog HDL language. It is simulated and synthesized using Xilinx ISE 7.1i software tool. Table 5 shows the comparison of device utilization and combinational path delay to compute circular convolution using KA and MKA. It is observed that circular convolution based on MKA requires least amount of area and path delay. Figure 9 shows the delay in computing convolution using two different algorithms and Figure 10 shows the resource utilization in terms of % of slices necessary for the implementation. In Spartan3E FPGA device family, computation of 8-bit circular convolution based on MKA is 26.5% faster than KA. It also consumes 61.7% less slices than existing KA based convolution.

VI. CONCLUSION

In this paper, modified Karatsuba multipliers for degree 3 and 7 polynomials has been implemented on FPGA platform. The device utilization and combinational path delay of MKM have been compared with standard 8×8 KM. It has been observed that the proposed multiplier has better timing performance than standard KM. In Spartan3E FPGA device, proposed multiplier needs 14.9% lesser delay than KM, and it also consumes 45.5% lesser slices compared to KM. The new architecture is very simple and easy. This feature is advantageous to have a suitable trade-offs between area and speed for implementing circular convolution algorithm in VLSI. In FPGA device family, computation of 8-bit circular convolution using MKA is 26.5% faster than KA. It also consumes 61.7% less slices than existing KA based convolution. MKM may also be used to design cryptosystems. Proposed multiplier is faster and hardware efficient compared to existing Karatsuba multiplier.

REFERENCES

[1] Z. J. Shi and H. Yun, " Software implementations of elliptic curve cryptography," *International Journal of Network Security*, vol. 7, no. 1, pp. 141-150, 2008.

[2] T. Zhang and K.K. Parhi, "Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials," *IEEE Trans. Computers*, vol. 50, no. 7, pp. 734-749, July 2001.

[3] C. Paar, P. Fleischmann, and P. Roeise, "Efficient Multiplier Architectures for Galois Fields GF(2^{4n})" , *IEEE Trans. Computers,* vol. 47, no. 2, pp. 162-170, Feb. 1998.

[4] C. A. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in GF(2^m)", *IEEE Transactions on Computers*,34(8):709- 717, Aug 1985.

[5] A. Reyhani-Masoleh and M.A. Hasan, "A New Construction of Massey- Omura Parallel Multiplier over GF(2^m)", *IEEE Trans. Computers*, vol. 51, no. 5, pp. 511-520, May 2002.

[6] Berlekamp, E. R., "Bit-Serial Reed-Solomon Encoder", *IEEE Trans. Inform. Theory,* Vol. IT-28, pp. 869-874 (1982).

[7] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers", in *Doklady Akad. Nauk SSSR*, vol. 145, no. 293-294, pp. 85, 1962.

[8] Koc, Cetin K; Erdem, Serdar S,"A Less Recursive Variant of Karatsuba-Ofman Algorithm for Multiplying Operands of Size a Power of Two", *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, 1063-1069,2003.

[9] Murat Cenk and Ferruh O¨ zbudak,"Improved Polynomial Multiplication Formulas over F_2 Using Chinese Remainder Theorem", *IEEE Transactions on Computers,* vol. 58, no. 4, pp. 572- 576, April 2009.

[10] Zhou, Gang; Michalik, Harald; Hinsenkamp, Laszlo, "Complexity Analysis and Efficient Implementations of Bit Parallel Finite Field Multipliers Based on Karatsuba-Ofman Algorithm on FPGAs", *IEEE Transactions on Very Large Scale Integration Systems,*18 (7), pp.1057-1066,2010.